ORBITAL DYNAMICS AND NUMERICAL N-BODY SIMULATIONS OF EXTRASOLAR MOONS AND GIANT PLANETS.

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Yu-Cian Hong

August 2019

ORBITAL DYNAMICS AND NUMERICAL N-BODY SIMULATIONS OF EXTRASOLAR MOONS AND GIANT PLANETS.

Yu-Cian Hong, Ph.D.

Cornell University 2019

This thesis work focuses on computational orbital dynamics of exomoons and exoplanets. Exomoons are highly sought-after astrobiological targets. Two candidates have been discovered to-date (Bennett et al., 2014, Teachey & Kipping, 2018). We developed the first N-body integrator that can handle exomoon orbits in close planet-planet interactions, for the following three projects. (1) Instability of moons around non-oblate planets associated with slowed nodal precession and resonances with stars.This work reversed the commonsensical notion that spinning giant planets should be oblate. Moons around spherical planets were destabilized by 3:2 and 1:1 resonance overlap or the chaotic zone around 1:1 resonance between the orbital precession of the moons and the star. Normally, the torque from planet oblateness keeps the orbit of close-in moons precess fast (period $\sim$ 7 yr for Io). Without planet oblateness, Io?s precession period is much longer ($\sim$ 10^4 yr), which allowed resonance with the star, thus the instability. Therefore, realistic treatment of planet oblateness is critical in moon dynamics. (2) Orbital stability of moons in planet-planet scattering. Planet-planet scattering is the best model to date for explaining the eccentricity distribution of exoplanets. Planets encounter each other closely, and moons are easily destabilized. The orbital evolution of planets also destabilizes moons via Kozai (highly inclined) perturbations and violation of Hill stability. Moons showed rich dynamical outcomes, including ejected free-floating exomoons, moon ex-

change between planets, moons turning to orbit the star, and moons orbiting ejected free-floating planets. Planets involved in planet-planet scattering develops high inclinations and high obliquities. Relevant instability effects for moons requires the code to address planet spin evolution. Planet-planet scattering is efficient at removing moons (80–90%). (3) Obliquity of extrasolar giant planets in planet-planet scattering. Planet-planet scattering can generate high obliquity giant planets and retrograde obliquity like Uranus. The close interaction during close encounters can't generate high obliquity (Brunini 2006, retracted), but the correspondence between the planets' spin and orbital precession rates can efficiently drive obliquity evolution (Storch et al., 2014). When planets are scattered close to the star, their obliquity evolves.

BIOGRAPHICAL SKETCH

Yu-Cian Hong was born in 1985 in Taichung, Taiwan. She attended National Taichung First Girls High School and majored in arts. She entered National Taiwan University in 2004 to major in European and American languages and literatures. From 2007, she also started a second major in physics. And in 2008, she did a 1 year overseas exchange program in the University of California Santa Barbara, and worked with experimentalist high energy physics groups there. In UCSB she found her true passion in scientific research.

From 2010 to 2012, she pursued master's degree in a joint Erasmus Mundus program in astronomy and astrophysics sponsored by the European Union. She received academic training in the University of Innsbruck, University of Rome Tor Vergata, University of Goettingen, and the University of Padova.

In 2012, she entered the Ph.D. program at Cornell University. At Cornell University, she pursued astrophysical research in computational orbital dynamics of extrasolar moons and extrasolar planets, as well as spectroscopic and direct imaging observations of planetary mass object with James Webb Space Telescope. Over the years in Ithaca, she developed passion in winter sports and nature photography.

Dedication to all that volunteered to support from near and far over the years of my PhD career. Special dedication to my father and mother who have made enormous effort and provided unconditional love to make me who I am today.

ACKNOWLEDGEMENTS

I am grateful to my mother Yue-Feng Lin and two loving brothers, who have provided unconditional support through my PhD. And I am grateful to my eternal life supporter, my late father Wen-Bin Hong, who is of course very proud of me at this moment of my life. Special gratitude to Hung-Jin Huang, my buddy in the pursuit of PhD, who has supported me like a family.

Of course a very special gratitude goes to my thesis advisor Prof. Jonathan Lunine, who has provided thorough support in academic life and work. His kindness and great leadership have created a safe and motivating work environment for me. His great scientific vision and skills as a leader in the field of planetary science has helped me in achieving creative and cutting-edge scientific results. Big thanks go to the great brains in the department, Prof. Phil Nicholson and Prof. Dong Lai, to whom I also seek science advice. I have also received enormous support from smart brains of Dr. Matt Tiscareno and Dr. Sean Raymond for my PhD works.

I have come to realize that any personal achievement in life is always founded on generous support from many others, and will pass it on to others.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Extrasolar moons and giant planets

1.1.1 Extrasolar moons

Extrasolar moons are highly sought-after astrobiological targets. Exomoon is also a cutting-edge research subject, with a lot of unknowns yet to be answered. Extrasolar moons could be interestingly diverse, habitable, or provide constraints on planet formation and evolution theory. The habitabilities of extrasolar moons have been extensively studied in various aspect for the potentially Earth-like worlds with large moons. There have also been continuous observational endeavors to discover extrasolar moons, with two candidates MOA-2011-BLG-262 and Kepler -1625b observed to date (Bennett et al., 2014, Teachey & Kipping, 2018). There are great future prospects of microlensing observations from WFIRST, down to Earth or Ganymede mass extrasolar moons, as well as free-floating moons down to Mars or Ganymede masses.

To address the existence of extrasolar moons, we need to study their formation and dynamical evolution. To date, extrasolar moon formation still need more research effort, although studies of moon forming disks and their observational aspects have been conducted (Szulágyi, 2017, Szulágyi et al., 2017, 2018). Since extrasolar planets are often found on extreme orbits, perturbations in the extrasolar systems can be much more robust. Therefore, studies of the dynamical evolution of extrasolar moons is important for ad-

dressing their existence and observational prospect. Previous works have explored satellite orbital stability in exoplanet systems in different dynamical settings - single and multivalent systems, and compact planetary systems citepbarnes,domingos,donnison,frouard,payne. In our study ,we will focus on the most dynamically violent and robust system - planet-planet scattering, and test whether moons will survive.

1.1.2 Planet-planet scattering in extrasolar giant planetary systems

Observations of many highly eccentric giant planets have inspired theoretical explorations of various planet formation and evolution models. The theoretical model that best fits the observed eccentricity distribution is planet-planet scattering (Chatterjee et al., 2008, Raymond et al., 2010) (Fig. 1.1). Mutual close encounters between planets produce dynamically excited systems, and cause ejection of planets into the interstellar space as free-floaters. Planet-planet scattering is also the best model for explaining Solar System formation in the Nice model. The Nice model invokes planet-planet scattering for the orbital evolution of giant planets, and had many successes in reproducing various features and populations in the Solar System,including the orbits of the giant planets, the asteroid belt, Jupiter Trojans, the Kuipter belt, the irregular moons, and the terrestrial planets, as well as the Late Heavy Bombardment (Brasser et al., 2009, Gomes et al., 2005, Levison et al., 2008, Morbidelli et al., 2010, 2005, Nesvorný & Vokrouhlický, 2009, Nesvorný et al., 2007, Tsiganis et al., 2005).

Planet-planet scattering starts with initially closely-packed planets. They

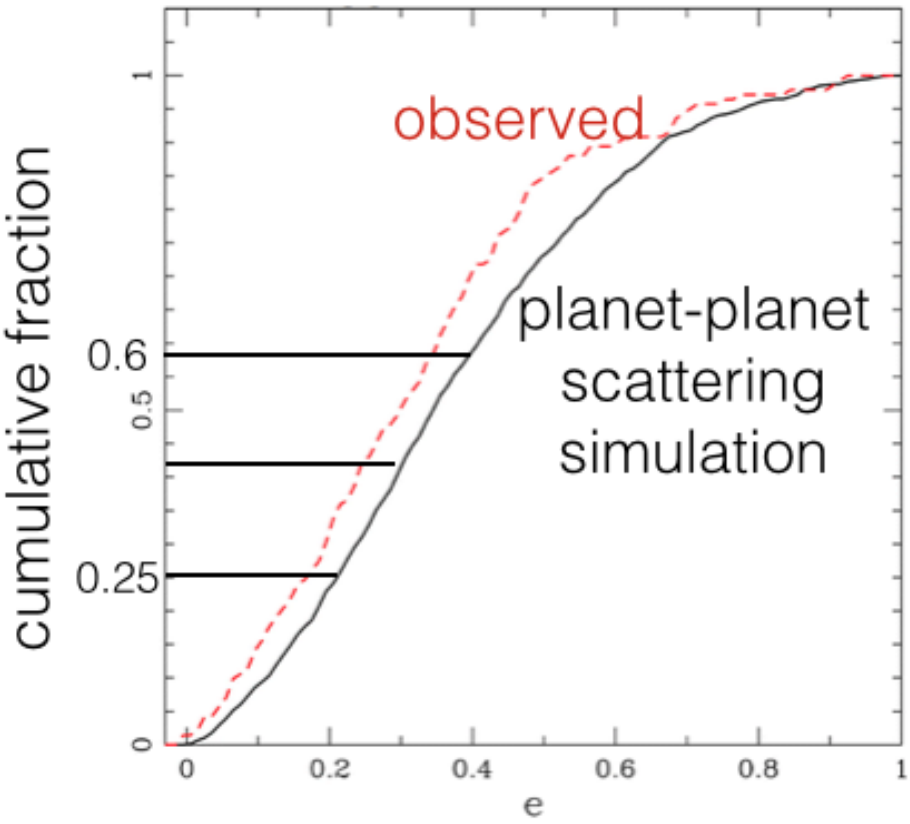

Figure 1.1 Observed and simulated distribution of eccentricities of extrasolar giant planets (Chatterjee et al., 2008).

experience strong mutual perturbations and deviate from Keplerian orbits. Through such evolution, planets can cross resonances and develop crossing orbits. During this unstable phase, planets experience close mutual flybys. The perturbation on planets come from the semi-secular perturbations between each other, as well as the close encounters. Both perturbations are significant in altering planetary orbits (Fig. 1.2). Planets experience radial migration, their eccentricities and inclinations evolve, as well as their spins. The system is restored to stability by removal of one or more of the planets in the system. The removed planet(s) carry away the excess angular momentum and leave the surviving planets on well-spaced orbits. Possible paths for the removal of planets include

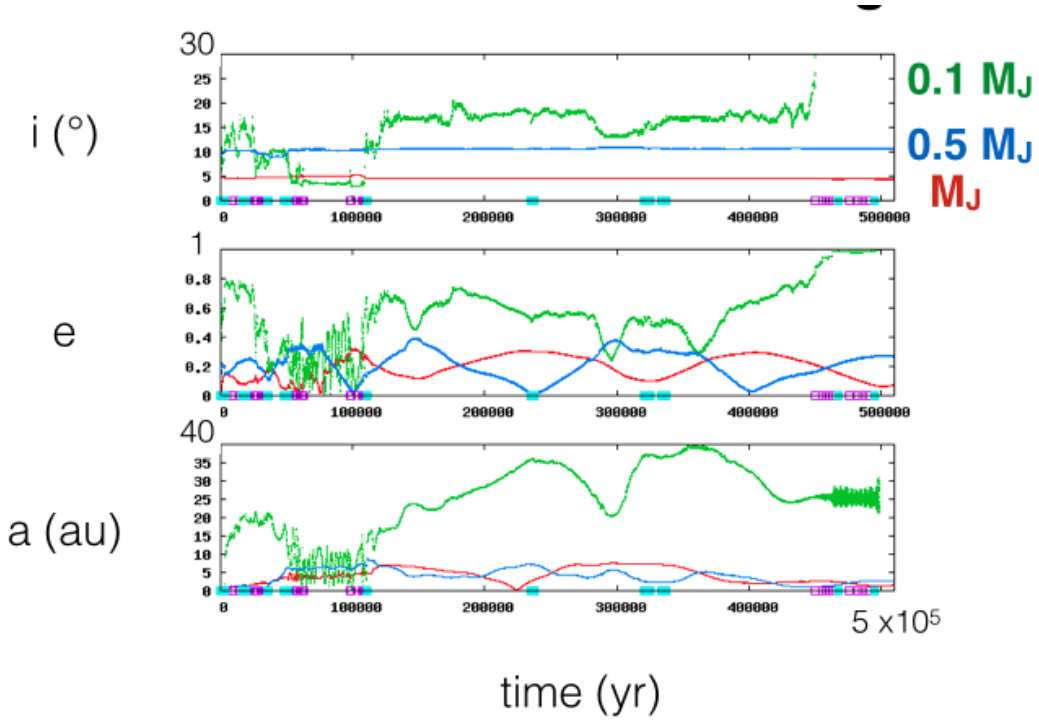

Figure 1.2 Oribtal evolution of three planet scattering.

ejection from the system, collision between planets, and collision with the star. Planet ejection links to the population of free-floating planets and their formation, although planet-planet scattering along can't fully account for the galactic population of free-floaters (Veras & Raymond, 2012), a couple of which have been discovered to-date (Lucas & Roche, 2000, Sumi et al., 2011, Zapatero Osorio et al., 2000). Collisions between planets can alter planet mass and spin, and collision with the star can change stellar composition and spin.

1.2 Overview of thesis

The work presented in this thesis focus on N-body simulations and orbital dynamics of extrasolar moons and extrasolar giant planets. For the purpose of

this work, we developed a symplectic N-body integrator specifically for moon dynamics in close planet-planet interactions. The code was modified from the Bulirsch-Stoer integrator in the MERCURY package (Chambers, 1999). Gravitational potential from planet oblateness is added to influence the dynamics of moons. The spin momentum of the planets are added as variables in the integration. The spin evolution follows a non-secular equation of motion, which is a suitable approach to problems where planet orbits change on a non-secular timescale. The modified code is a set of ~ 5000 lines Fortran code.

This thesis consists of three parts: (1) Orbital instability of moons around non-oblate planets associated with slowed nodal precession and resonances with stars. (2) Orbital dynamics of moons in extrasolar giant planet-planet scattering. (3) Obliquity of extrasolar giant planets - the missing parameter space in both theoretical and observational literature - in planet-planet scattering.

In chapter 2, we tests the orbital stability of moons in an unusual dynamical setting - moons around spherical giant planets. We reversed the commonsensical notion that spinning giant planets should be oblate. Moons around spherical planets were destabilized by 3:2 and 1:1 resonance overlap or the chaotic zone around the 1:1 resonance between the orbital precession of the moons and the star. Normally, the torque from planet oblateness makes the orbits of close-in moons precess fast (period $\sim$7 yr for Io). Without planet oblateness, Io's precession period is much longer ($\sim 10^4$yr), which allows resonance with the star, thus the instability. Therefore, realistic treatment of planet oblateness is critical in moon dynamics.

In chapter 3, we study the orbital dynamics of moons of extrasolar giant planets in planet-planet scattering. Moons showed rich dynamical out-

comes, including ejected free-floating exomoons, moon exchange between planets, moons turning to orbit the star, and moons orbiting ejected free-floating planets. Planets involved in planet-planet scattering develop high inclinations and high obliquities. Relevant instability effects for moons require the code to address planet spin evolution. Planet-planet scattering is efficient at removing moons (80-90 $\%$). The majority of moons are lost to collision with larger bodies and ejection as free-floaters. However, close-in moons like the Galilean moons can be stable. Simulations predict abundant free-floating exomoons in our Galaxy (0.01-1 per star). Microlensing observation with WFIRST can detect free-floating moons down to Mars mass (Bennett et al., 2018) or Ganymede mass (Johnson et al., in prep).

In chapter 4, We study the obliquity evolution of planets in planet-planet scattering. Planet-planet scattering can generate high obliquity (angle between planet's spin and orbital axis) giant planets. For obliquity excitation to happen, spin and orbital precession periods need to be similar. If the spin of the planet precesses much slower than the orbit of the planet, it will stably precess the orbital axis; if the spin precesses much faster than the orbit, it will follow the orbit normal well (Storch et al., 2014). In both cases, the obliquity remains constant. When two conditions are satisfied, planet obliquity can evolve 1) correspondence between the planet's spin and orbital precession periods, and (2) planets are scattered close to the star to receive enough torque. Preliminary results show that planets can develop obliquities higher than 40° via planet-planet scattering, and even retrograde obliquities like Uranus. Doppler imaging provides opportunities of observing planet obliquities (Vanderburg et al., 2018).

CHAPTER 2

ORBITAL DYNAMICS OF CLOSE-IN EXTRASOLAR MOONS AROUND NON-OBLATE GIANT PLANET

Yu-Cian Hong, Matthew S. Tiscareno, Philip D. Nicholson, Jonathan I. Lunine

Monthly Notices of the Royal Astronomical Society, 449:828, 2015

2.1 Introduction

Giant planets in our Solar System all have extensive natural satellite systems, which has led to extensive studies on whether exoplanets likely also harbor satellites. Exomoons could be interestingly diverse, habitable, or provide constraints on planet formation and evolution theory. Since the discovery of exoplanets, the habitability of their moons has been studied considering different aspects such as stellar illumination, stellar irradiation, satellite atmosphere, planet magnetic field, etc. (Heller, 2012, Heller & Barnes, 2013, Heller & Zuluaga, 2013, Kaltenegger, 2010, Williams et al., 1997). Exomoons have some advantages over exoplanets on the extent of habitable orbital configurations. The possibilities of them 1) orbiting around the confirmed giant planets in the classical liquid water habitable zone determined by stellar illumination, planetary greenhouse effect and others (Kasting et al., 1993), and 2) retaining the right temperature for stable liquid water outside the classical habitable zone by tidal heating (Heller & Barnes, 2013, Reynolds et al., 1987, Scharf, 2006) could make them desirable targets for Earth-like habitability studies.

Various techniques have been studied for detection of exomoons. 1) transit

timing variation of the moon-hosting close-in planets (Agol et al., 2005, Holman & Murray, 2005, Kipping, 2009, Kipping et al., 2012, Sartoretti & Schneider, 1999, Simon et al., 2007) can detect moons down to 0.2 Earth size with the Kepler mission (Kipping et al., 2009). 2) Microlensing detection of exomoons can reach down to 0.01 Earth masses (Bennett & Rhie, 1996, 2002, Han, 2008, Han & Han, 2002). It is better suited to detect moons that orbit planets distant from the parent star. A sub-Earth mass exomoon candidate has been found orbiting a free-floating giant planet via this technique (Bennett et al., 2014). 3) Direct imaging can detect bright, tidally heated exomoons down to 1 Earth size with temperatures above 300K and 600K with JWST and Warm Spitzer (Peters & Turner, 2013).

However, questions arise on whether exomoons exist because we understand very little about their formation and evolution processes. Sources of perturbation to their orbital evolution are very different from within the Solar System. Unlike the rather circular, co-planar, and well-separated orbits of Solar System planets, the orbits of many exoplanets are eccentric and they may also be mutually inclined as generally predicted by the planet-planet scattering model (Chatterjee et al., 2008, Jurić & Tremaine, 2008, Marzari & Weidenschilling, 2002, Raymond et al., 2010). Some multiplanet systems are found to be relatively compact, and many exoplanets are on extremely close orbits around the parent star. Competition between perturbations to their orbits and the host planet's gravitational attraction determine whether moons can remain on stable orbits.

Previous works have explored satellite orbital stability in exoplanet systems in different dynamical settings. Some have focused on systematical numerical studies of single planet systems (Barnes & O'Brien, 2002, Domingos et al., 2006,

Donnison, 2010, Holman & Wiegert, 1999), in which the satellite removal processes could be stellar tidal stripping, violation of the Hill stability criterion, etc. Recent studies proceeded to multiple planet systems, where planetary perturbation on the moons becomes very important because orbital spacings between planets are compact or they experience planetary close encounters (Frouard & Yokoyama, 2013, Gong et al., 2013, Hong et al., 2012, Nesvorný et al., 2007, Payne et al., 2013). Planet oblateness was considered as negligible in the scope of the above mentioned works.

This work will show that, in multiplanet systems where the orbits of planets are mutually inclined, close-in satellites situated within a critial planetocentric distance where perturbation to the orbit is usually dominated by planet oblateness (Nicholson et al., 2008) can become dynamically unstable when planet oblateness is neglected. Non-coplanarity provides a path for the occurence of the nodal precession of the planets and satellites, and as will be shown in section 4, the approach of secular resonances of the nodal precession rates of the innermost satellites with the host planet (or the central star in the host planet's frame) may be the cause for instability. Such instability needs to be considered in setting the satellite-hosting planet's oblateness in order to obtain reasonable numerical results.

We simulate systems with two planets mutually inclined by 10°. The host planet is treated as spherical in simulation set 1 and as oblate in simulation set 2 for comparison. As will be shown in section 3, in simulation set 1, satellites with small planetocentric distances gain high inclination or leave planet-bound orbits, while distant satellites stay on low inclinations and relatively unperturbed orbits. By contrast, such unusual dynamical phenomena for the close-in satel-

lites disappear in simulations with an oblate host planet, now that the gravitational potential in the region close to the planet becomes dominated by the J_2 moment term.

2.2 Simulation setting

This work uses the N-body symplectic integrator Mercury (Chambers, 1999). All simulations use the Bulirsch-Stoer algorithm, the most accurate for simulating bodies that perturb each other closely, though the slowest in the package, in order to exclude integration error as a cause for the problematic orbits of satellites in this work. Unless otherwise specified, the simulations are configured as follows : 1) the integration error limit in orbital energy and angular momentum is 10^{-12}, 2) the integration time-step is 0.1 days, in order to accurately integrate satellites with orbital periods as short as a few days, and 3) the simulation duration is 1 million years.

Simulations test the orbital stability of primordial satellites in non-coplanar two-planet systems. All simulations consist of a central star, two giant planets and their satellites. The scene is set after the final stage of satellite formation and after the circumplanetary disk has dissipated. The mass and radius of the central star are about that of the Sun – 1 Solar mass and 0.0046AU. The masses and densities of the giant planets resemble those of Jupiter – 9.548×10^{-4} Solar mass and 1.3 g/cm^3. The inner planet is always at 5 AU from the star, and the orbits of both planets are nearly circular and mutually inclined by 10°. The satellites are massless test particles. They orbit around the inner planet (hereafter the host planet) at a range of planetocentric semi-major axes a_s = 0.008 – 0.25 Hill radii

(hereafter R_H)[1], or a_s = 0.002731 – 0.08534 AU. The innermost satellite approximates Io's position around Jupiter and lies beyond 3 times the Roche limit for satellites with a density higher than 0.5 g/cm^3, so that it's safe to neglect tidal mass loss or tidal disintegration (Guillochon et al., 2011). The outermost satellite is well within the stability limit [2] of $\sim$ 0.5 R_H (Domingos et al., 2006), as we have no interest in satellites that become unstable due to violation of the Hill stability criterion. Most of the simulations have 3 satellites at a_s = 0.008, 0.06, and 0.25 R_H, unless specified. The discussion section gives some special focus on the satellite at $a_s = 0.008\, R_H$, hereafter called the innermost satellite. The satellites are on initially circular ($e < 0.0001$), prograde orbits, and are co-planar with the host planet's initial orbital and equatorial plane. The gravitational interactions between satellites are not included in any simulations. All bodies are non-spinning.

In total there are 4 sets of simulations.

Simulation set 1 has 25 simulations and the only variable between simulations is the semi-major axis of the exterior planet (hereafter the perturber), which spans the range a_p = 6.602 – 12.612 AU, corresponding to planet-planet orbital period ratios $\frac{P_2}{P_1}$ = 1.517 – 4.01, and mutual Hill radii of 3.21 – 4.01. Due to our interest in stable planetary systems, perturbers in all simulations initially lie outside the boundary of global chaos. (Veras & Armitage, 2004) If the planet separations were inside the global chaos limit, they would be bound to undergo gravitational scattering. The dynamical instability criterion associated

[1] In the circular restriced 3-body problem, the Hill radius of a planet marks the boundary within which the gravitational attraction of the planet dominates the star's tidal field. $R_{Hill} = a_p \left(\frac{m_p}{m_*}\right)^{\frac{1}{3}}$, where m_p is planet mass and m_* the central star's mass.

[2] This is the stability limit for prograde satellites in single planet systems with the planet and satellites on nearly circular orbits.

with non-coplanar systems will be further discussed in section 3.

Unlike what is assumed in simulation set 1, the majority of real giant planets may be oblate. Giant planets are non-rigid bodies that contain spin angular momentum inherited from the orbital angular momentum of the smaller bodies in the circumstellar disk that form them, so they experience rotational flattening. A planet distorted in shape modifies the gravitational potential around itself[3].

Simulation set 2 is a duplicate of simulation set 1, except that the host planet is oblate, with a J_2 moment of 0.0147, equal to that of Jupiter, with its equatorial bulge fixed in the initial reference plane. The difference between simulation sets 1 and 2 leads to results that address the main theme of this work: the host planet's J_2 moment plays a crucial role in the stability of close-in satellites – that the absence of it can lead to instability or even loss of satellites.

Simulation set 3 is a duplicate of simulation set 1, except that perturber-moon interaction is turned off. This set tests whether the perturber directly causes the counter-intuitive behavior of the innermost satellites.

Simulation set 4 takes the simulation with $\frac{P_2}{P_1}$ = 4.01 from simulation set 1, and changes the integration error limit and time-step to 10^{-13} and 0.001 days. Simulation set 4 double checks whether the problematic orbits of satellites in simulation set 1 could be due to the lack of integration accuracy.

[3]Assuming the planet has uniform density and is an ellipse with axial symmetry, the modification can be approximated by a term with the second order Legendre polynomial with its factor the J_2 moment, $V(r,\theta) = -\frac{Gm}{r}[1 - J_2 \cdot (\frac{R}{r})^2 \cdot P_2(cos\theta)]$ (Murray & Dermott, 1999). G is the gravitational constant, m the planet mass, R the planet's equatorial radius, r the planetocentric distance, θ the angle from the principle axis, and J_2 a dimensionless constant.

2.3 Simulation results

All output orbital elements of planets in this section are calculated in the star-centered frame, and those of the satellites in the host-planet-centered frame. Different planes of reference are used where appropriate, and they are 1) the initial orbital plane of the host planet and satellites, which is identical to the host planet's equatorial plane, 2) the host planet's precessing orbital plane, and 3) the system's invariable plane determined by the total angular momentum.

2.3.1 Planet orbits

Starting with a 10° mutual inclination and nearly circular orbits, the planets in simulation set 1 and 2 are initially well-separated enough so their mutual perturbations cause little change in semi-major axes and eccentricities over 1 million years, and their mutual inclination remains close to constant at all times.

However, as seen from the initial reference plane, the inclinations of the two planets oscillate sinusoidally with a constant period but different amplitudes [4]. The direction of the host planet's equatorial bulge, which is fixed on the initial reference plane, in turn oscillates sinusoidally with the same period and angle as the host planet's inclination oscillation when seen from the host planet's orbital plane. This setting could give rise to unphysical motion for close-in satellites since in reality the planet's obliquity should evolve as its orbital inclination

[4] This is as predicted by the Laplace-Lagrange theory of secular perturbation. The range of their inclination can be approximated by: $0 \leq I_1(t) \leq \frac{2I_0}{1+\sqrt{\frac{a_1}{a_2}}}$, and $I_0 \times \frac{1-\sqrt{\frac{a_1}{a_2}}}{1+\sqrt{\frac{a_1}{a_2}}} \leq I_2(t) \leq I_0$ (Veras & Armitage, 2004). The suffix 0 represents initial values at t = 0, suffix 1 represents the inner planet (host planet), and suffix 2 represents the outer planet (perturber).

changes; however, it does not affect the simulation results within this work since the innermost satellites stay on the planet's equatorial plane, as they should be around planets with low obliquity. (Goldreich, 1965) The maximum changes in semi-major axes of the planets in all simulations of set 1 and 2 are under 2%, in eccentricities under 0.05, and in inclinations under 0.4°, using the host planet's orbital plane as reference. Dynamical instability does not occur in these selected simulations, and their orbits deviate from each other little enough for the purpose of comparing satellite orbits between the 2 simulation sets.(figure 2.1).

As a side note, in a few simulations, planet orbits evolve chaotically. Non-coplanarity allows the planets with separations larger than the global chaos limit to undergo gravitational scattering (orbit crossing) or have their orbital elements evolve significantly. These simulations have their initial planet separations close to the boundary of global chaos or to first- or second-order resonances (Veras & Armitage, 2004). We do not include those simulations in the discussion but only focus on the stable ones, so that the effect of planet oblateness plays the major factor in 2 comparable sets of simulations.

2.3.2 Satellite Orbits

In simulation set 1, the absence of the host planet's oblateness perturbation causes the innermost satellites to undergo unusual orbits, as will be illustrated in the sub-sections below. However, by treating the host planet as oblate in simulation set 2, moon orbits become tame and stable, and their motion is predicted to a good approximation by the Laplace theory.

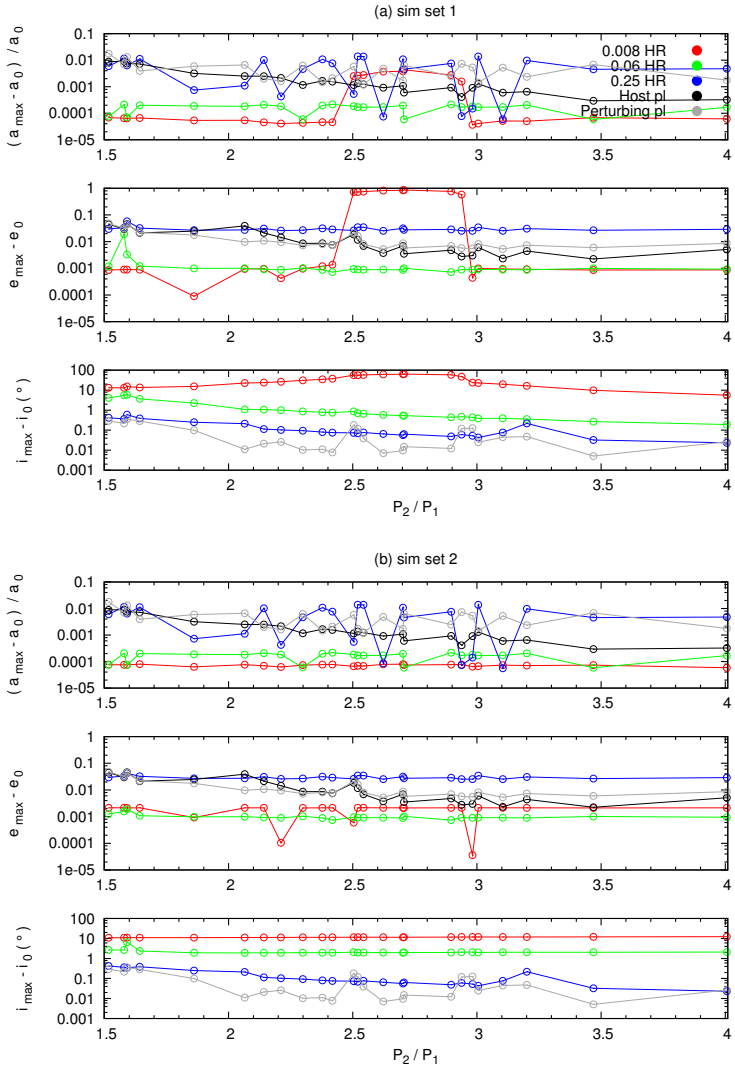

Figure 2.1 The maximum change in the orbital elements (semi-major axis a, eccentricity e, and inclination i) of planets and satellites throughout each simulation in set 1 and 2. The reference plane for both simulation sets is the host planet's precessing orbital plane. The legend box shows the color representation of planets, and satellites by their semi-major axes. Planets in both simulation sets experience very mild changes in orbits, and the maximum deviation in both simulation sets are somewhat similar. Like the planets, change in semi-major axes of the satellites are mild. However, in simulation set 1, there is a trend for the innermost satellites to experience higher inclinations than the outer ones. The innermost satellites in the range $\frac{P_2}{P_1}$ = 2.50 – 2.94 even reached inclinations beyond 40 ° and eccentricities above 0.5. On the other hand, the more distant satellites at a_s = 0.06 and 0.25 R_H experience much milder changes. In simulation set 2, the trend for distant satellites to stay close to the host planet's orbital plane is the same, but the misbehaved innermost satellites now find a home in the host planet's equatorial plane; the innermost and outer satellites' eccentricities now both remain small, and all satellites are stable.

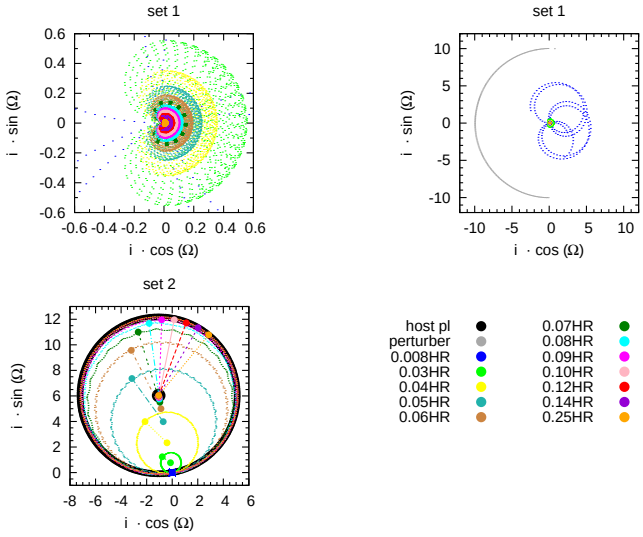

Figure 2.2 The evolution of the angular momentum axes of planets and 12 satellites in the simulation with $\frac{P_2}{P_1}$ = 4.01 in both simulation set 1 and 2, in the first 229,090 years. The legend box shows color representation of planets, and satellites by their semi-major axes. The host planet's plot is exaggerated in thickness for clarity. The reference plane for simulation set 1 is the host planet's precessing orbital plane, and for simulation set 2 the host planet's equatorial plane. Simulation set 1 is plotted twice with different scales. Distance from the origin represents inclination, which increases with decreasing satellite semi-major axes. Each satellite appears to precess around some plane that itself is precessing around the host planet. On the other hand, satellites in simulation set 2 all precess around an average plane of their own (the center of each circle), which moves from the host planet's equatorial plane to its orbital plane as the satellites' semi-major axes increase. The radius of the circle is their inclination relative to the average plane. The innermost satellite only stays on the host planet's equatorial plane, while the outer most ones stay close to the host planet's orbital plane.

Satellite orbital plane

In the upper panel of figure 2.2 is a selected simulation from set 1 with $\frac{P_2}{P_1}$ = 4.01, where satellites with smaller semi-major axes are more inclined than those with larger ones, and their orbital normal, the angular momentum axis determined by inclinations and nodes, precesses around a plane that itself is precessing around the host planet's orbital normal, which sits at the origin of the plot.

On the other hand, satellites with larger semi-major axes tend to stay close to the host planet's orbital plane, or in other words, the time evolution of their angular momentum axes follow that of the host planet closely.

When this simulation is rerun in set 2 with an oblate host planet, the satellites become well-behaved. As shown in the lower panel of figure 2.2, the innermost satellites stay on the host planet's equatorial plane at all times, and satellites at different semi-major axes orbit around their individual Laplace planes, which transition from the host planet's equatorial plane to the host planet's orbital plane as a_s increases. No satellites have gone onto a plane outside this region, as do the innermost satellites in simulation set 1. Because the perturber's gravitational influence is negligible compared with that of the star, as is shown in figure 2.2 , the satellites' motion appears to conform with the 200-year-old theory of Laplace surface (Tremaine et al., 2009).

High inclination and instability

As shown in figure 2.1, simulation set 1 exhibits an overall trend for close-in satellites to stay farther away from the host planet's orbital plane than more distant ones in both stable and unstable cases. The innermost satellites' maximum inclination varies from 5.6° to 64.1°. In each simulation with $\frac{P_2}{P_1}$ = 2.50 - 2.94, the innermost satellite's maximum inclinations are higher than 47°, and eccentricities above 0.58, whereas in other simulations of set 1 inclinations are under 40° and eccentricities are negligibly small. On the other hand, more distant satellites expriences much milder changes in orbits. Figure 2.3 illustrates how the innermost satellites' inclinations and eccentricities evolve through time. The innermost satellite in $\frac{P_2}{P_1}$ = 2.70 is lost to collision with the host planet due to its high

eccentricity (> 0.8). The reason for the satellites' eccentricity to rise might be that the increased inclinations cause some perturbative terms to become large. In the lower panel, for simulations outside the region $\frac{P_2}{P_1}$ = 2.50 - 2.94, satellite orbits are much less disturbed and instability doesn't occur, although most innermost satellites are still more inclined than the outer ones; also, the peak inclinations of moons drop as $\frac{P_2}{P_1}$ moves away from the unstable region, and all moons have eccentricities under 0.002. As a side note, though instability occurs in simulation set 1, all satellites have fractional changes in semi-major axis under the order of 0.01. This points to secular effects as the culprit, as they are well known to alter eccentricities and inclinations while doing no work on the semi-major axes.

By contrast, in simulation set 2, as shown in figure 2.1 (b), the innermost satellites' maximum changes in eccentricity are all under 0.005, and most of them have zero inclination relative to the the host planet's equatorial plane at all times, with only two reaching $i \sim 0.02^\circ$. All satellites, including outer ones, stay on stable orbits around their host planet, and the change in orbital elements of all satellites in the simulation set remains reasonably mild throughout.

The motion of the satellites in simulation set 2 are more physical and stable compared with simulation set 1. It will lead to significant differences in important metrics such as exomoon survival rate. As the setting in simulation set 2 is also more realistic, we conclude that adding the host planet's equatorial perturbation is a necessary measure for simulating exomoons in systems with mutually inclined planets.

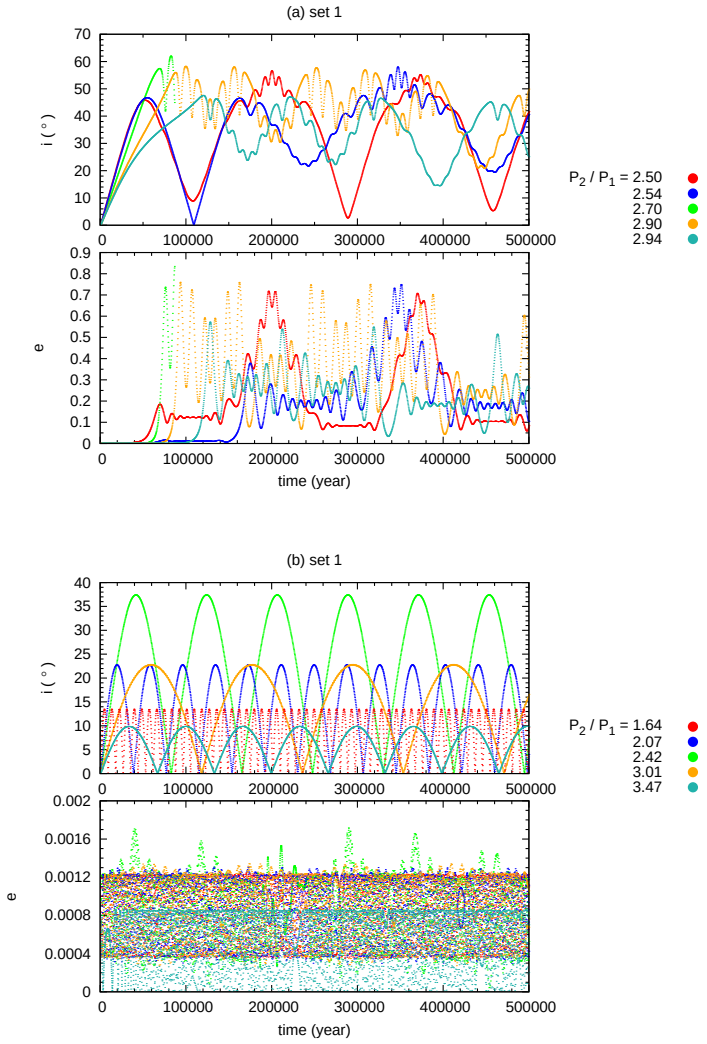

Figure 2.3 Time evolution of the innermost satellites' (a_s = 0.008 R_H) orbital elements in several simulations in simulation set 1. The reference plane is the host planet's precessing orbital plane. The legend box shows color representation of satellites by the simulation they belong to. The evolution of semi-major axis is not shown because the change is very mild ($< 1\%$). In (a) the innermost satellites in $\frac{P_2}{P_1}$ = 2.50 – 2.94 undergo rather disturbed orbits in inclinations and eccentricities. All of them have reached i $> 40\,^\circ$ and most of them have e > 0.7. The innermost satellite in $\frac{P_2}{P_1}$ = 2.70 has the highest e (> 0.8) and i ($> 60\,^\circ$), and is lost to collision to the host planet due to its high e. (b) shows the orbits of the innermost satellites in $\frac{P_2}{P_1}$ outside the range 2.50 – 2.94. Their inclinations are milder ($< 40\,^\circ$), although still higher than the outer moons, and much less disturbed. Their eccentricities are also very small (< 0.002). The peak inclination drops as $\frac{P_2}{P_1}$ moves away from the region 2.50 – 2.94.

2.4 Potential causes for instability

2.4.1 Direct perturbation by perturber

Simulation set 3 tests the hypothesis that the inner moons' behavior in set 1 is caused by the direct gravitational effects of the perturbing planet, which may compete with the direct effect of the star (which is also inclined, due to the perturbing planet's effect on the host planet) such that the star's effects dominate for most satellites but fall off with decreasing satellite semi-major axis more steeply than do the direct effects of the perturbing planet, thus leading to abnormal behavior by the innermost satellites.

Simulation set 3 is a duplicate of simulation set 1, except that perturber-moon interaction is turned off. The unusual behavior of the satellites will disappear if the above hypothesis is right. The planets' orbits do not differ from set 1 and 2, and the moons' behaviors are qualitatively the same as in set 1. The evolution of the angular momentum axes of moons in $\frac{P_2}{P_1}$ = 4.01 in set 3 is almost the same as in set 1 (upper panels of figure 2.2). Also, as shown in figure 2.4, inner moons have higher inclinations than outer moons. The innermost moons in $\frac{P_2}{P_1}$ = 2.50 - 2.94 have inclinations above 45.8° and eccentricities mostly above 0.65, whereas in the rest of the simulations their inclinations are below 40° and eccentricities mostly below 0.04 but one at e = 0.11. Many of the moons' maximum changes in eccentricities are at least an order of magnitude higher than in set 1. The higher eccentricities in set 3 appear to be a result of forced perturbation by the eccentric star. By comparison, in set 1, the direct influence of the perturbing planet could give the satellites enough free eccentricity to avoid such forced perturbation. Despite this difference, simulation set 3 demonstrates that

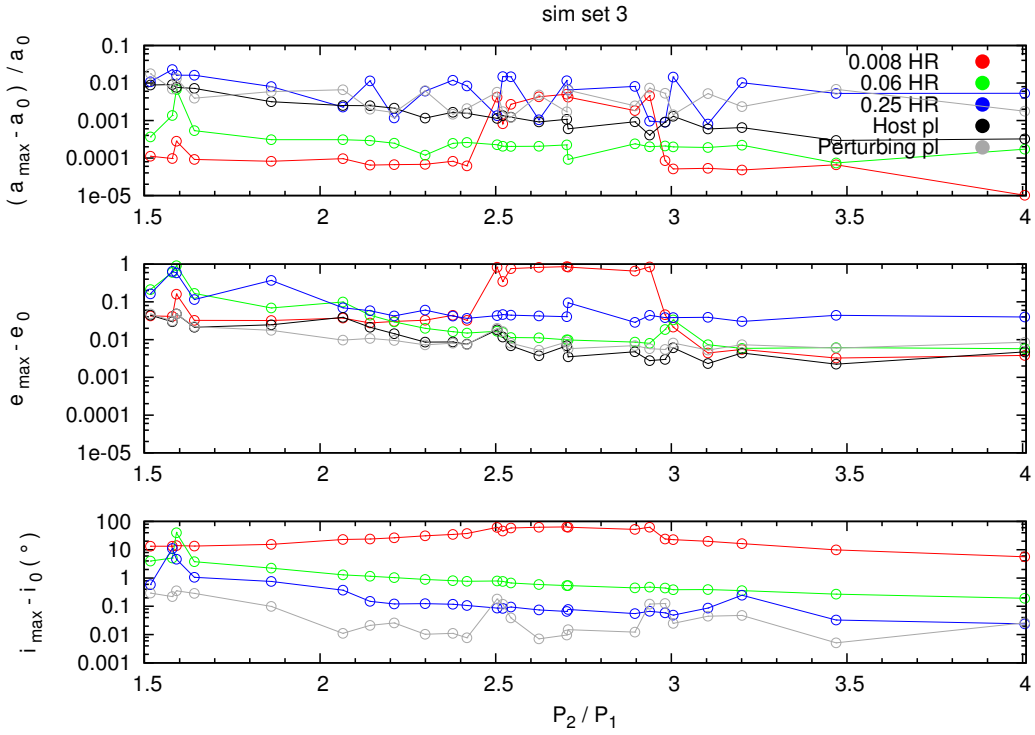

Figure 2.4 The maximum change in the orbital elements of planets and satellites throughout each simulation in simulation set 3. The reference plane is the host planet's precessing orbital plane. Planet orbits resemble that of set 1, and satellites demonstrate similar behavior as in figure 2.1 (a), although many have eccentricities ~ 1 order of magnitude higher.

direct perturbation from the perturber is not the major cause for the satellites' behavior in set 1. Instead, the culprit may involve the precession of the host planet's orbital plane (or, from the host planet's point of view, the star's orbital plane) indirectly caused by the perturber.

2.4.2 Secular resonance

Around an oblate planet, the critical distance separates the region where planet oblateness and stellar perturbation dominates. It is located where the two per-

turbations are equal.

$$a_{crit} = \left(2\, J_2\, R_p^2\, a_p^3\, \frac{m_p}{m_*} \right)^{\frac{1}{5}}, \tag{2.1}$$

where J_2 is the quadruple moment of the planet, R_p the planet's radius, a_p the planet's distance from the star, m_p the planet's mass, and m_* the star's mass. (Kinoshita & Nakai, 1991) The host planet's a_{crit} in set 2 is 0.015 AU ($0.044\, R_H$). The fact that the innermost satellite in set 1 lies within this distance but with the planet oblateness missing could potentially give rise to instability associated with secular resonances.

When a satellite's nodal precessional motion resonates with one of the system's eigenfrequencies, it is said to be in secular resonance. Secular resonance of nodes can increase the inclination of a body. Examples in the Solar System are asteroids and trojans that resonate with ν_{16}, which roughly equals Jupiter's nodal precession rate.

The precessional motion of the satellites in this work is made possible by the mutual inclination of the planets, which left the initial orbital plane after time t = 0. Due to the lack of planetary bulge, the satellites' nodal precession is dictated by stellar perturbation. The nodal precession rate is given by

$$-\frac{3}{4}\, \frac{n_*^2}{n_s}\, cos\, i\,, \tag{2.2}$$

where n_* is the mean motion of the star and n_s that of the satellite, and i is relative to the host planet's equatorial plane. Satellites with smaller orbital distances precess slower than distant ones, and the calculated period of one precession cycle is $\sim 3.6 \times 10^4$ years for the innermost satellite. In the simulations, the innermost satellite's period of nodal precession is 3.7×10^4 years at $\frac{P_2}{P_1}$ = 1.52, and it slowly increases as $\frac{P_2}{P_1}$ increase, probably due to an increase in i. On the other hand, the nodal precession period of the host planet is 6.4×10^3 years at $\frac{P_2}{P_1}$

= 1.52, much shorter and increases much faster than the innermost satellite as $\frac{P_2}{P_1}$ increases (fig. 2.5), since the major source of perturbation to the satellite is the central star and to the host planet is the perturbing planet. This allows the host planet and the satellite to hit several resonances at different $\frac{P_2}{P_1}$. As shown in figure 2.5, instability happens as they hit the 3:2 and upon entering the 1:1 secular resonance ($\frac{P_2}{P_1}$ = 2.50 - 2.94), and the 1:1 resonance as seen from the invariable plane ($\frac{P_2}{P_1} > 2.94$) appears to have a stabilizing effect. The fact that unstable cases all lies in between the 3:2 and 1:1 secular resonances could point to resonance overlap as the cause for instability. [5]

In figure 2.6 are the phase space plots of the innermost moons. The unstable cases show traces of being affected by the 1:1 secular resonance, and they happen in between circulating (upper left) and librating (lower right) motions. The transition through the separatrices into the 1:1 resonance could likely cause the satellite's inclination to experience large oscillations. And the stability of the innermost satellites in simulation set 2 could be due to the enhanced rate of their nodal precession by the host planet's J_2. In 2 out of 25 simulations where the innermost satellites' precession around the host planet's spin axis occurrs at intervals when it acquires a tiny inclination of 0.02°, the period is on the order of 10 years.

[5] In perturbed Hamiltonian systems, the separatrices on the phase plane are not perfect thin lines but disarrayed layers. As the perturbative term is being cranked up, the upper and lower separatrices of two largest resonances spread toward each other, and as the critical perturbation is reached, the separatrices touch, and then systems originally locked in a isolated resonance would transition through the chaotic zone weaved by the layered separatrices and experience chaos. This is called Chirikov's resonance overlap criterion. (Chirikov, 1979)

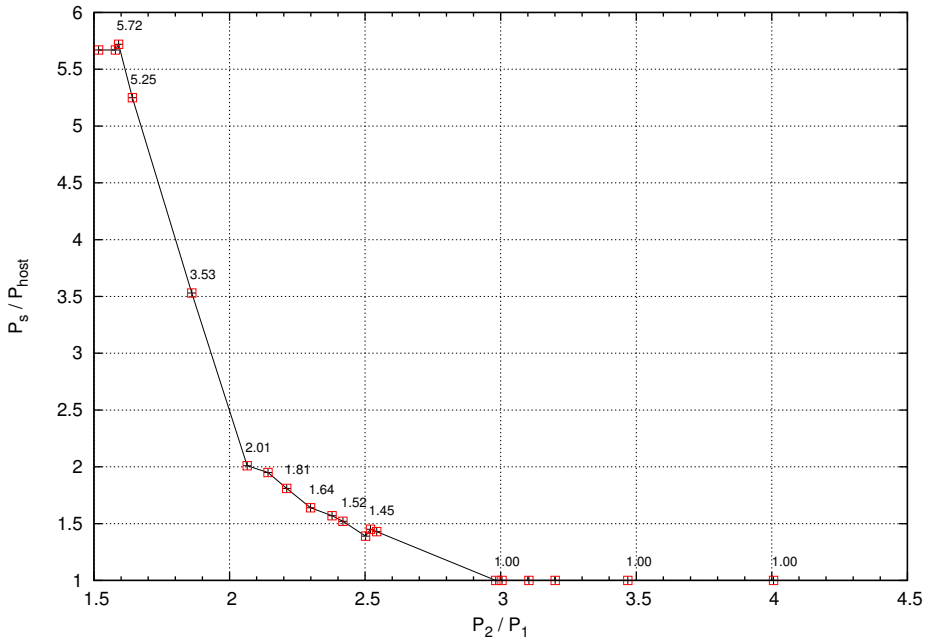

Figure 2.5 Planet period ratio vs. ratio of satellite and star nodal precession rate. The vertical axis represents the period ratio of nodal precession of the innermost satellite versus that of the host planet (or the star in the host planet's frame), and on the horizontal axis is the planet's orbital period ratio. The reference plane is the system's invariable plane. At small $\frac{P_2}{P_1}$, the host planet precesses much faster than the satellite, but its precession rate decreases much faster than the satellite with increasing $\frac{P_2}{P_1}$, therefore the former eventually hit the 2:1, 3:2, and then enter the 1:1 resonance with the latter. As they enter the 3:2 resonance, the effect of the approach of the 1:1 resonance starts to destabilize the satellites. The instability could be associated with the chaotic zone in the vicinity of the 1:1 secular resonance or in the overlap region of the 3:2 and 1:1 resonances.

2.4.3 Numerical error

Simulation set 4 double checks that the behavior of the innermost satellites in set 1 are not attributable to numerical error. The simulation with $\frac{P_2}{P_1}$ = 4.01 in simulation set 1 were rerun with a smaller integration error limit of 10^{-13}, and a shorter integration time step of 0.001 days. The resulting orbits of the innermost satellites resemble those in simulation set 1. In addition, among all 3 simulation sets, the maximum simulation error in energy dE/E is 2.58×10^{-8} and in angular

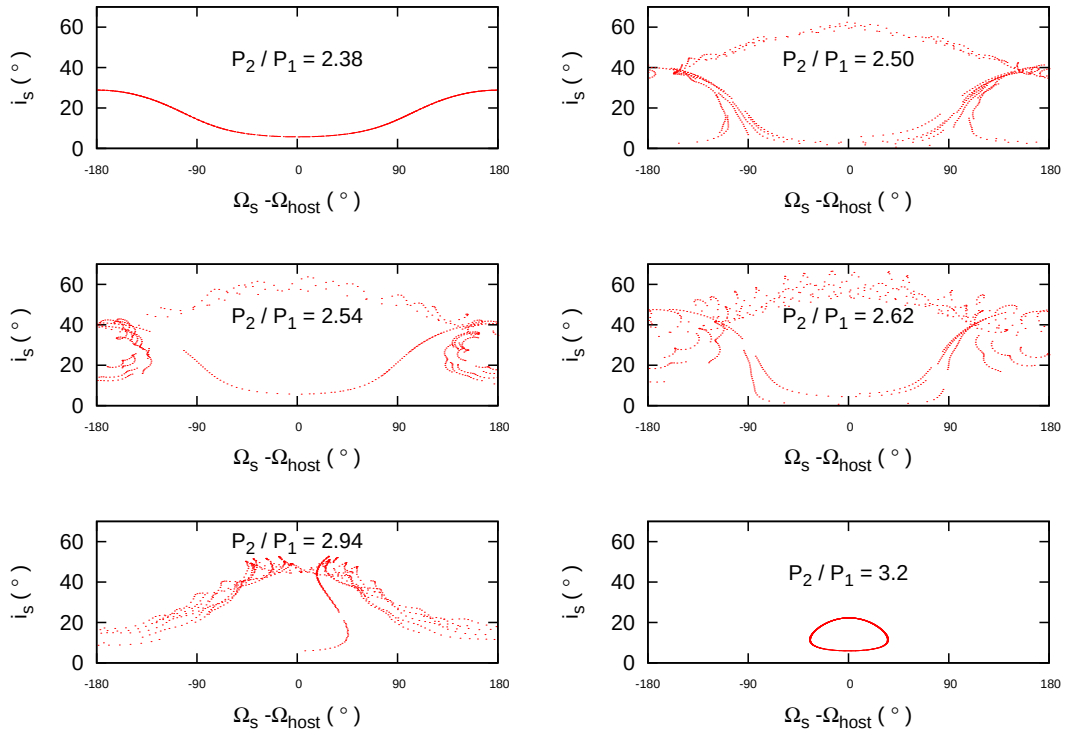

Figure 2.6 Phase space plots of the innermost satellites. The orbital elements are calculated on the system's invariable plane. The upper left plot is a case before entering the 3:2 secular resonance showing circulating motion and the lower right a case in the 1:1 resonance showing librating motion. The middle 4 plots are cases in the overlap region of the 3:2 and 1:1 resonance. This could imply that the transition into the 1:1 resonance is responsible for the satellite's high inclination.

momentum dL/L is 7.90×10^{-9}. Therefore, we do not attribute the innermost satellites' behavior to numerical error.

2.5 Conclusions

The close-in satellites within the critial distance where planet oblateness usually dominates demonstrate very different dynamics from their Solar System counterparts when the planetary bulge is absent. The planetary bulge enhances the

nodal precession rate of satellites, and without it, the nodal precession rate of a satellite can be slow enough to hit secular resonances with the host planet, which could potentially cause the inclination to increase. These unusual dynamical effects disappear when the host planet is oblate, as we would expect. As a side note, the type of instability demonstrated in this work could not occur for perfectly co-planar systems, and is not relevant for satellites beyond a_{crit}.

Numerical modeling often needs to make assumptions, and it is common to neglect factors not immediately interesting or relevant. However, when simulating exomoon orbits, we may encounter dynamical systems that have planets on mutually inclined orbits. Some known exoplanet systems are on more compact and eccentric orbits, and they may acquire high mutual inclination as they interact. They may also encounter each other extremely closely. In those systems, perturbations may often be greater than in this work and without planet oblateness the system could encounter a similar effect seen in this work.

CHAPTER 3

ORBITAL DYNAMICS OF MOONS OF EXTRASOLAR GIANT PLANETS IN PLANET-PLANET SCATTERING

Yu-Cian Hong, Sean N. Raymond, Philip D. Nicholson, and Jonathan I. Lunine

The Astrophysical Journal, 852:85, 2018

3.1 Introduction

Given that each of the Solar system's giant planets hosts at least one large natural satellite, the presence of moons is anticipated around giant exoplanets. The potentially diverse environments on exomoons, and the clues they may provide to planet formation models makes them subjects worthy of research.

The potential habitability of exomoons is affected by a number of processes, including atmospheric dynamics, stellar illumination, tidal heating, planetary magnetic fields, orbital configurations, etc (Dobos et al., 2017, Forgan & Kipping, 2013, Heller, 2012, Heller & Barnes, 2013, 2015, Heller & Zuluaga, 2013, Hinkel & Kane, 2013, Kaltenegger, 2010, Tinney et al., 2011, Williams et al., 1997). Exomoons have more paths to habitable configurations than planets. Besides being in the liquid water habitable zone of the star (Kasting et al., 1993), effects such as tidal heating provide an opportunity for moons outside the stellar habitable zone to be heated (Dobos et al., 2017, Heller & Zuluaga, 2013, Reynolds et al., 1987, Scharf, 2006).

Several techniques appear capable of detecting exomoons (see Section 6 of Heller et al. (2014) for a review). The transit technique has the prospect of ob-

serving sub-Earth sized moons (Agol et al., 2005, Heller, 2014, Holman & Murray, 2005, Kipping, 2009, Kipping et al., 2012, 2009, 2015, Sartoretti & Schneider, 1999, Simon et al., 2007), and has discovered a Neptune-sized exomoon candidate (Teachey & Kipping, 2018). Microlensing can detect moons down to 0.01 Earth masses (Bennett & Rhie, 1996, 2002, Han, 2008, Han & Han, 2002) and has discovered a sub-Earth mass exomoon candidate around a free-floating planet (Bennett et al., 2014). Its broader sensitivity with heliocentric distances may yield a better chance of exomoon detection. Direct imaging may be able to detect bright, tidally heated exomoons (Peters & Turner, 2013).

The frequency of the occurence of exomoons depends on their formation and dynamical evolution. The Hill stability criterion, Roche limit, stellar tidal stripping, tidal decay, planet migration, direct planet perturbation, and planetary close encounters are among the mechanisms that can destabilize the orbits of exomoons (Barnes & O'Brien, 2002, Domingos et al., 2006, Donnison, 2010, Frouard & Yokoyama, 2013, Holman & Wiegert, 1999, Kane, 2017, Namouni, 2010, Payne et al., 2013, Sasaki et al., 2012, Spalding et al., 2016). The stability of distant satellites ($a_s > 0.1\ R_H$), which would be classified as irregular satellites in the Solar System, in planet-planet scattering, are investigated by Gong et al. (2013) and Nesvorný et al. (2007).

In this paper we investigate the orbital dynamical behavior and the stability of primordial satellites as close-in as Io to Jupiter ($\sim 0.01\ R_H$), in scenarios where planets gravitationally scatter off each other. Planet-planet scattering has been considered as the most viable candidate mechanism for explaining the prevalence of eccentric orbits of extra-solar planets and reproduced well their observed eccentricity distribution (Adams & Laughlin, 2003, Chatterjee et al.,

2008, Ford & Rasio, 2008, Jurić & Tremaine, 2008, Lin & Ida, 1997, Marzari & Weidenschilling, 2002, Rasio & Ford, 1996, Raymond et al., 2010, Weidenschilling & Marzari, 1996). Therefore, it is important to test the orbital stability of planetary satellites in this context. In the planet-planet scattering scenario, planets are hypothesized to form closely packed, then they perturb each other and eventually enter an instability phase when they undergo orbit crossing and close encounters. The instability is ended by the removal of some planets in the system by ejection or collision. In the instability phase, the planetary satellites experience strong gravitational perturbations from the perturbing planets. Various sources of perturbations also affect the satellites, such as secular perturbations from the perturbing planets (as opposed to the host planet of the satellites) , and stellar perturbations through the change of the host planet's spin and orbits in the instability phase. The orbital parameter space of moons in this work covers close-in regions where planet oblateness plays a major role in moon stability, as an already well-established fact, and the absence of planet oblateness will cause unrealistic orbital instability effect (Hong et al., 2015). This work also simulates moons within and beyond the critical semi-major axis (0.04 Hill radii for Jupiter) of the planets where planet spin can affect moon stability (Tremaine et al., 2009), and moons up to 0.35 Hill radii where prograde moons can be stable.

Section 2 studies the inner working of planetary close encounters and discusses factors related to close encounters that affect moon stability, by simulating single planet mutual flyby events. Section 3 discusses the orbital evolution of moons under various sources of perturbations, moon survival rate and its relevant factors, and the dynamical outcome of moons in planet-planet scattering in full simulations of length 1-100 million years.

3.2 Numerical method

This work uses the N-body symplectic integrator *Mercury* (Chambers, 1999). All simulations use the Bulirsch-Stoer algorithm, the most accurate for simulating bodies that perturb each other closely, though the slowest in the package, in order to assure integration accuracy. The conservative version of Bulirsch-Stoer integrator is used, which is appropriate in cases where the force does not depend on velocity. The spin and oblateness of the planet are adapted into the code in order to correctly simulate the stability of planetary satellites.

3.2.1 Oblateness of planet

The planets are simulated as oblate by an addition of the quadrupole moment (J_2) term to the gravitational force between point mass bodies. This treatment ensures that the orbits of moons within the critical orbital distance from a planet are simulated correctly and that unexpected instability will not occur while the planet keeps the close-in moons' orbital angular momentum precessing rapidly (Hong et al., 2015). Numerically this is done by adding a customized force term that accounts for the quadrupole moment contribution in the gravitational attraction of a planet to other planets and satellites in *Mercury*. Since the spin of the planet is allowed to change, the direction of the force from the planet's bulge follows the spin evolution of the planet. However, the code assumes that the spin of gas giant planets can react to external torques instantaneously during a planetary close encounter, whereas in reality the timescale for a fluid body to react to torques might be longer than the close encounter time.

3.2.2 Spin evolution of planet

The spin of an oblate planet evolves under the influence of the central body and other planetary bodies, especially during a close encounter between planets. In the planet-planet scattering scenario, the spin of a planet can often evolve dramatically, in which the stability of its moons needs to be tested (Tremaine et al., 2009). In this work, the spin evolution model for the planet uses an instantaneous equation of motion, in order to correctly simulate the effect of planetary close encounters on the spin of a planet. The equation of motion for the spin evolution is derived in the following paragraphs.

An extended mass receives different amounts of gravitational attraction from a distant point mass at different radial distances within its interior, which results in a torque on the extended mass, causing its spin to evolve. The force from a point mass M on an extended mass m is

$$\vec{F} = \frac{-G\,M\,m}{r^2}\{\hat{r} - 3\,J_2\,(\frac{R}{r})^2\,[(5(\hat{r}\cdot\hat{s})^2 - 1)\,\hat{r} - 2\,(\hat{r}\cdot\hat{s})\,\hat{s}] + \cdots\}, \tag{3.1}$$

where G is the gravitational constant, r is the distance between the centers of m and M and $\hat{r}$ points from m to M, R is the radius of m, and $\hat{s}$ the normalized spin vector of m (Hilton, 1991). The torque per moment of inertia is

$$\frac{\vec{r}\times\vec{F}}{I} = \frac{d}{dt}\,(\Omega\,\hat{s})\,, \tag{3.2}$$

where Ω is the rotation period of the planet, which for simplicity is always assumed to be constant and equal to 10 hours, the same as for Jupiter. The time

evolution of the spin components of an extended mass can thus be described in the following equation

$$\frac{d\hat{s}}{dt} = \frac{-3\,G\,M}{r^3}\frac{J_2}{\lambda\,\Omega}\left(\hat{r}\cdot\hat{s}\right)\left(\hat{r}\times\hat{s}\right), \tag{3.3}$$

where M is the distant perturbing mass,$\hat{r}$ is the unit vector of displacement between the perturbing mass and extended mass, λ is the normalized moment of inertia. λ is taken to be 0.25, close to that of Jupiter, in all simulations. The above equation is derived assuming an instantaneous force and torque within which no variables are averaged over time. It is therefore suitable for the planet-planet scattering case, where planets sometimes interact extremely closely $(ie. \sim 0.001AU)$, thereafter over a brief period of time $(ie. \sim 1yr)$. In this work, the back reaction of planet oblateness on the orbits of the star and other planets are neglected. In comparison, as a note, equations of spin evolution for secular problems such as the obliquity evolution of Mars require the orbit to stay unchanged at least over the timescale of the body's orbital period:

$$\frac{d\hat{s}}{dt} = \left(\frac{-3}{2}\frac{n^2}{\Omega}J_2\lambda\right)\left(\hat{l}\cdot\hat{s}\right)\left(\hat{l}\times\hat{s}\right) \tag{3.4}$$

(Bills, 1990, 2005, Ward, 1973, 1974). The above equation uses averaged quantities such as $\hat{l}$, which is the angular momentum averaged over an orbital period. During a close encounter, in particular, such an assumption lacks accuracy because the small timescale of the variation of perturbation is sometimes comparable to the orbital period of the binary planet pair.

Eq. (4.1), with its dependence on the body's position relative to other bodies, is plugged into the Bulirsch-Stoer integrator alongside the integration for posi-

tions and velocities, and is updated once per time step. The validity of the integrator is tested with the past obliquity evolution of Mars. The initial conditions are taken from Quinn et al. (1991) at epoch JD 2433280.5, with the obliquity of Mars $= 25^{\circ}$. The simulation is integrated backwards and produces results with close resemblance to Ward (1973).

Unless otherwise specified, the initial settings of the simulations are configured as follows : 1) the integration error limit in orbital energy and angular momentum is 10^{-12}, 2) the integration time-step is 0.1 days, in order to accurately integrate satellites with orbital periods as short as a few days like the Galilean satellites.

3.3 Result: single close encounter

We start by considering the effect of a single planet-planet encounter on the stability of one planet's moons.

3.3.1 Simulation setting

In each simulation of simulation set 1, the system contains a Sun-like star, a Jupiter-mass planet that hosts moons, and a perturbing planet. The host planet is at 5 AU on a circular orbit. Moons are massless test particles placed between 0.01 – 0.35 Hill radii (thereafter R_H) from the host planet on circular and coplanar orbits. Four moons are spaced out evenly in angular position at each of the 10 planetocentric semi-major axes. The planets are set up to undergo close encounters (within each other's Hill sphere) by the following. The perturbing

planet is initially placed at 1.2 Hill radii from the host planet. The parameters of the perturbing planet are the only variants across different simulations in set 1. The perturbing planet's mass is equal to 0.1, 0.5, 1.0, or 2.0 M_J (Jupiter mass). Its starting velocity relative to the host planet (V_{rel}) ranges from 0.001 to 0.009 AU / day, and the direction of its velocity vector is restricted to lie within 60° from the host planet's velocity vector. The range of V_{rel} is determined from all close encounters in a subset of planet-planet scattering simulations for 10^6 yr. The perturber's starting impact parameter (b) is 0.005 - 0.1 AU (host planet radius = 0.00047 AU). The impact parameter is set as a 3-dimensional sphere of vectors surrounding the center of the host planet. All simulations last for 10 years. Simulations in which planets collide with each other or in which the perturber starts with e > 1 are excluded. In most of the close encounters during the instability period, the latter configurations are rare.

3.3.2 Results

The minimum close encounter distance d_{min} plays a major role in determining moon stability. From the initial conditions, it is mainly b that determines d_{min} and thus moon stability. V_{rel} plays a minor role because the escape velocity of the host planet (0.035 AU / day) is 1–2 orders of magnitude ($\sim$ 35–350 times) greater than V_{rel}. Figure 3.1 shows the stability boundary of moons around host planets that experience close encounters with different closest approach distances. The stability boundary is set where at least 2 moons on the same planetocentric distance survive with e $\leqslant$ 0.5, because they are anticipated to have high probabilities to survive a full simulation. As will be discussed in section 3.4, 80% of the surviving moons have eccentricities under 0.5. The sta-

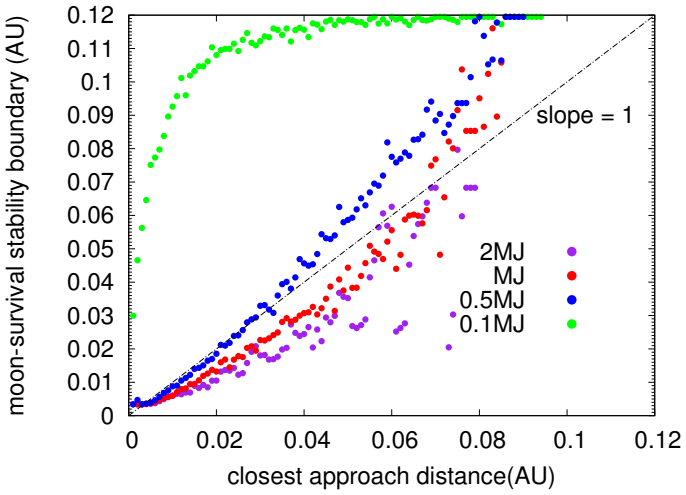

Figure 3.1 Planet minimum close encounter distance vs. moon stability boundary. The stability boundary is set where the most distant surviving moon with $e < 0.5$ is located around the planet. Dots with different colors represent simulations with different perturber mass as shown in the legend. Each dot is obtained from taking averages of simulations with the same d_{min}. The trend in the plots with perturber mass 0.5 – 2 M_J indicates that d_{min} plays the most important role in determining the moon stability boundary, and the different slopes of each curve corresponding to different perturber masses shows that perturber mass is a secondary factor.

bility boundary for each d_{min} is an average taken from all the host planets with the same d_{min} and with a perturber having the same mass. The larger d_{min} is, the larger the stability boundary. For simulations with perturber mass 0.5–$2M_J$, the stability limits in figure 3.1 are positively correlated with d_{min} in the curves, which demonstrates that d_{min} plays the most important role. The slope of the curves is determined by the perturber mass, demonstrating that the perturber mass plays the second most important role in determining the stability boundary of a planet. The slope is roughly linear when the perturber mass is less than twice that of the host planet, but when the perturber mass reaches twice that of the host planet, the linear relation starts to fail.

Host planets with lower-mass perturbers have larger stability boundaries.

Simulations with a 0.5 M_J perturber have stability boundaries close to or greater than d_{min}, and simulations with a 0.1 M_J perturber have much larger stability boundaries than d_{min}. For simulations with larger perturbers (M_J and 2 M_J), the stability boundary is 0.6–0.8 times d_{min} for $d_{min} < 0.05AU$. Except for simulations with a 2 M_J perturber, there exists a boundary in d_{min} beyond which almost all moons from 0.01–$0.35R_H (\sim 0.003$–$0.12AU)$ are stable. For simulations with a 0.1/0.5/1 M_J perturber, when d_{min} is greater than $\sim 0.05/0.1/0.1$ AU, the stability boundary lies close to 0.35 R_H.

Regarding the moon capture rate, like the moon survival rate, the most important determining factor is d_{min}, and the perturber mass the second. Figure 3.2 shows the average number of moons captured by the perturber from the host planet in simulations with different d_{min}'s. A perturber that has come closer to the host planet is able to pass by more moons, and because capture of moons by the perturber requires the perturber to be close to moons (fig. 3.3(b)), a smaller d_{min} yields a higher capture rate. For planet masses (0.5–$2M_J$) comparable to the host planet, the capture rate is roughly inversely-proportional to d_{min}. Larger perturber mass slightly enhances the capture rate. As seen from figure 3.2, the moon capture rate is only significant (> 0.1) for d_{min} under $\sim 0.03AU$ for simulations with 0.5–$2M_J$ perturbers. 0.1 M_J perturbers have nearly no capability of capturing moons from a 1 M_J host planet during their encounter, as seen from the average number of captures, but in a few cases the capture rate can reach 1–5%.

Figure 3.3 shows how the close encounter geometry can affect the dynamical outcome of moons. The scenario contains an M_J host planet and an M_J perturber. d_{min} for this encounter event is 0.01 AU, shorter than the initial semi-

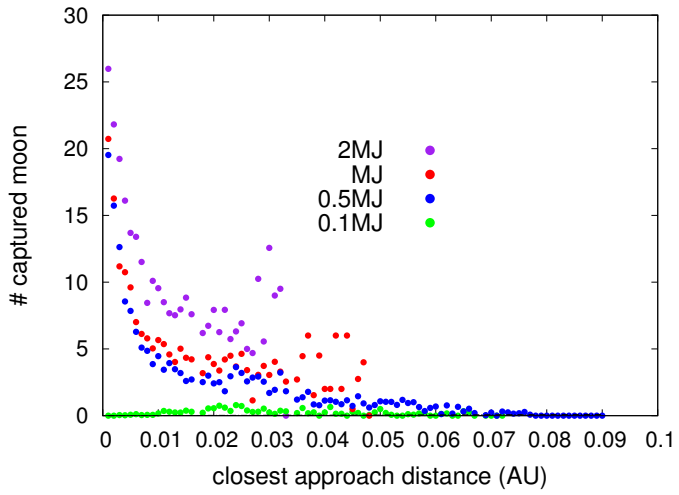

Figure 3.2 Minimum close encounter distance vs. average number of moon captures per planet per close encounter. Figure legend as in figure 3.1.

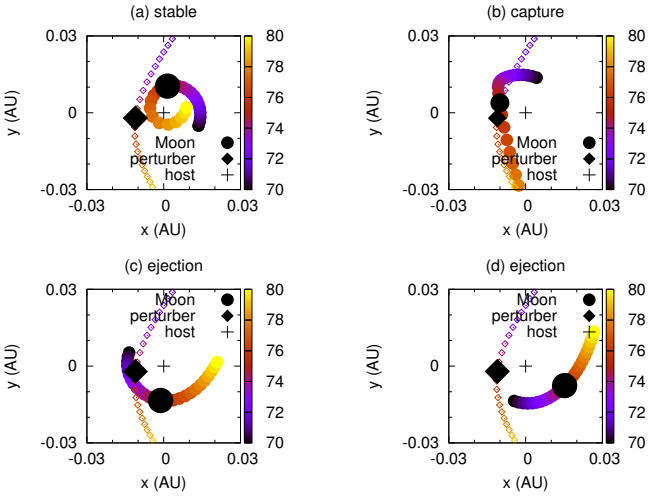

Figure 3.3 Close encounter geomentry vs. moon dynamical outcome. The same planetary close encounter event is split into four subfigures. Each subfigure features a different moon in the system. In each subfigure, the host planet is represented by the cross sign at the origin, the trajectory of the perturbing planet is represented by empty circles, and the trajectory of the moon represented by empty squares. Their trajectories change in color from purple to yellow as time evolves, as is labeled by the vertical color axis (time in days). The solid black dots represent the location of the perturber and the moon at the time of closest approach. See the main text for further simulation details.

major axis of the four moons (0.014 AU / 0.042 R_H) with their angular positions evenly spaced out. Despite having the same initial orbital radius, the moons had a range of dynamical outcomes. In (a), the moon is dragged in the forward direction and changes its orbit but stays stable. In (b), the moon heads toward the planet and is able to reach it with the right timing and configuration from the back of the planet for a smooth transfer onto a stable orbit around the perturber to happen. In (c) and (d), the moons are dragged in the backward direction by the planet and thus open up their orbits to become unbound. In summary, moons on the same orbits can end up with different dynamical outcomes based on their relative distance from the perturber and also the alignment of acceleration from the perturber with the velocity vector of the moon at the time of close encounter.

3.4 Result: full integration

3.4.1 Simulation setting

In set 2, we simulate the dynamical evolution of exomoons in a more general context and over longer timescales. Each simulation consists of a central star, three giant planets, and moons orbiting all the planets. All bodies in the system are assumed to be fully formed by the time the simulation starts, and the protoplanetary disk is absent.

The star has solar mass and solar radius. Each simulation includes three giant planets, and the combinations of planet masses (chosen from 0.1, 0.3, 0.5, or 1 M_J) for the three planets in each simulation are sampled thoroughly and with

equal frequency in the final statistics (2 simulations for systems with equal mass planets and 1 simulation for each of the other combinations of planet masses, totaling a set of 68 simulations). Simulations where planet collisions occur are rerun, because planetary collisions could introduce complexities to moon survival. This procedure may introduce bias into the final statistics because roughly a quarter of three-planet systems have planet collisions (Raymond et al., 2010), and many of the planetary collisions happen early on. This leaves the third planet – which does not participate in the collision nor very much in the close encounters – with a very high moon survival rate. The innermost giant planet is placed at 5 AU from the central star, and all planets are separated by 3.5 to 4.5 mutual Hill radii, in order for instability to set in within a reasonable timescale (Chambers et al., 1996, Marzari & Weidenschilling, 2002). In addition, they are given initial eccentricities randomly sampled from 0.02 – 0.1, in order to shift the first onset of instability to an earlier time. To avoid immediate collisions between the giant planets upon orbit crossings, they are also given a small mutual inclination of 0.01°. The giant planets are treated realistically as oblate spheroids, with their quadrupole moment $J_2 = 0.0147$ equal to that of Jupiter, and the spin of the planet is allowed to evolve in this simulation set.

Each of the 3 giant planets in a simulation initially has 10 satellites. Each satellite is placed at a different planetocentric orbital distance from $0.01R_H$ to $0.35R_H$ (0.0016 - 0.24 AU for the entire simulation set) on circular orbits. The inner boundary of $0.01R_H$ is close to Io's orbital distance from Jupiter, a distance far enough away from the Roche limit to avoid tidal mass loss and disintegration due to the tidal field of the planet (Guillochon et al., 2011). The outer boundary lies within 0.5 R_H, the Hill stability limit for prograde satellites (Domingos et al., 2006). The moons are initially on circular and nearly co-planar

orbits (0 – 0.02°) with their host planet's orbital plane and equatorial plane. All simulations in set 2 are integrated to 1 million years. The final results discussed below are drawn from this full set of simulations unless specified. A subset of the above simulations are further integrated up to 10 million years. The energy error $\frac{dE}{E}$ accumulated in the simulation due to the integrator is always under 10^{-4}, which is an adequate threshold for multi-planet systems (Barnes & Quinn, 2004, Raymond et al., 2010). A couple of the simulations with $\frac{dE}{E} \sim 10^{-4}$ in the above subsets are rerun or ruled out. All simulations in set 2 are integrated to 100 million years for testing the stability of "moons" that end up orbiting the star. This set has a higher rate of exceeding the threshold $\frac{dE}{E}$ ($\sim 20\%$) but those cases are ruled out. All moons are massless test particles that do not perturb each other or the planets.

3.4.2 Moon orbital evolution under different perturbations

During the instability phase of the system, when planets sometimes encounter each other closely, and have strong secular interactions, moons' orbits evolve under the influence of various perturbations. The most influential and in most cases the strongest perturbation comes from the close approach of the perturbing planet to moons. In addition, as the planets' and moons' orbits evolve through the instability phase, the stability of moons are affected. Changes in the host planet's semi-major axis, eccentricity and inclination affect the size of the planet's Hill sphere; changes in the moon's own orbits also cause them to become Hill unstable. A rise in the planets' and moons' inclination can put moons under Kozai-like perturbations. The spin evolution of planets could also affect the stability of moons that are close to the critical semi-major axis, as will

be further explained below. Mutual planet secular perturbations also perturb moons. Different types of orbital instability may also have been involved in destabilizing the moons.

Direct effect of planetary close approach

During a close encounter, planets and moons have strong gravitational interactions over a brief period of time, typically on the order of 10^{-1} to 1 years. Therefore, the time evolution of orbital elements of planets and moons experience an instantaneous change at close encounters, as shown in fig. 3.4. The planet's spin axis also experiences a small sudden change at a close encounter, causing a slight change in the satellite's plane of nodal precession. But the inclination of the moon from the equatorial plane often changes more dramatically during a close encounter. Depending on the geometry of the encounter, their semi-major axis, eccentricity and inclination may increase or decrease. Although the close encounter geometry is random, the overall eccentricity of the moon tends to increase than to decrease. Shown in figure 3.4 is a typical case of how the moons' eccentricity evolves during the instability phase. At a very close encounter that is strong enough to perturb most of the moons, their eccentricities increase instantaneously by a large amount. Subsequent milder encounters are able to perturb stable moons that are already eccentric, so then the eccentricity increases or decreases depending on the encounter geometry. Sometimes more encounters push the moons onto highly eccentric orbits or destabilize them.

Figure 3.4 also demonstrates how a very close encounter determines the moon's orbit and dynamical outcome. If a perturbing planet has a mass comparable to the host planet, and it approaches the host planet more closely than

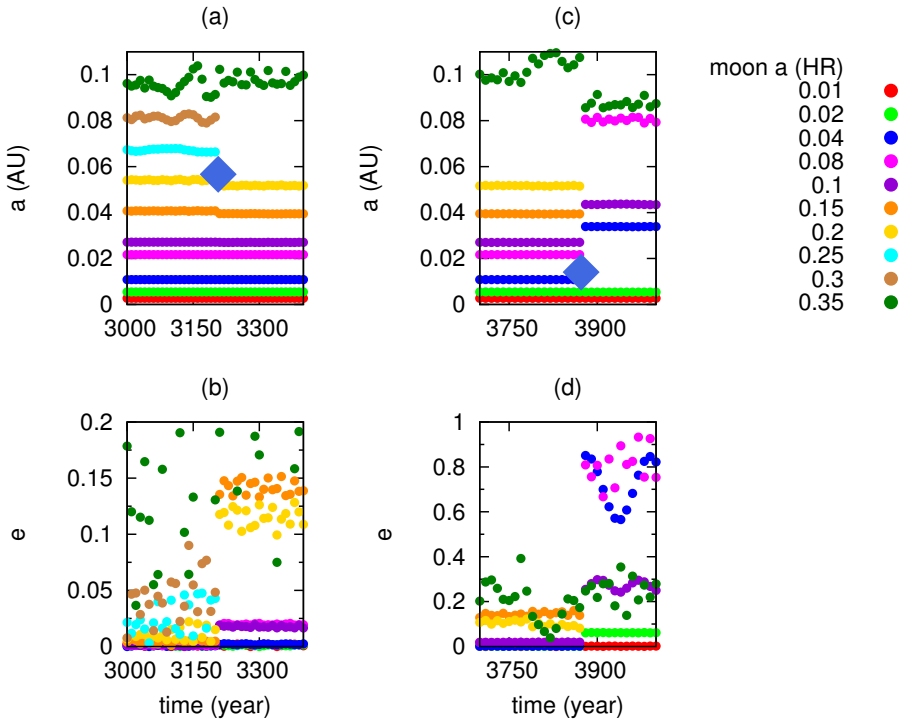

Figure 3.4 Evolution of moon a, e early in two early close encounter events. (a) and (b) features the 1st close encounter, and (c) and (d) feature the 2nd one in the same simulation. The two events are two consecutive effective close encounters. Each colored curve represents a moon with their initial orbital distance in R_H labeled in the legend. All moons orbit the same 0.5 M_J host planet. In (a) and (b), at $\sim$ 3200 yr, a 0.1 M_J perturbing planet (blue diamond) makes a very close approach to the moon hosting planet with $d_{min} = 0.057AU$. (c) and (b) features a subsequent close encounter at $\sim$ 3850 yr with $d_{min} = 0.014AU$.

some of the moons, moons exterior to its closest approach distance all become destabilized. If the perturbing planet is much less massive than the host planet (ie. at least 3 times smaller, as in fig. 3.4), not all moons become destabilized. This demonstrates again how d_{min} determines the moon stability limit as in section 3.3, and also why close-in moons tend to have higher survival rates. In figure 3.4 (a) and (b), moons interior to the closest approach distance exhibit a trend of eccentricity growth correlated to semi-major axes. In figure 3.4 (c) and (d), the close encounter further randomized the outcome of the previous one. The 0.1 M_J perturbing planet pumps up the eccentricity of the majority of moons ex-

terior to and close to its closest approach distance, and destabilizes two of the most exterior moons. Close encounter geometry and moon semi-major axes have similar effects as in the previous close encounter. In between the two featured close encounter events, there are two close encounters with much larger d_{min}, far away from the moons, and together with the small mass of the perturbing planet, they cause little perturbation on moons. The more distant moons, due to their closer distance to the perturbing planet, tend to become more eccentric. In the case that not all moons are destroyed, moon survival is not as dependent on semi-major axis; the close encounter geometry (or the relative distance and velocity between the moon and the perturber) plays a more important role than in the previous case, although the strong gravitational binding of closer-in moons could still give them a somewhat better chance of survival. If the closest approach distance is exterior to the moons, chances are higher the moons could survive but those within the sphere of the perturber's gravitational influence will become more eccentric, as a general trend, the larger their semi-major axes. However, the encounter geomentry adds some uncertainties. In summary, all factors of survival come into play together, but on longer timescales than shown in figure 3.4, the seemingly important role of minimum close encounter distance is smeared out.

Destabilized moons can collide with the host planet due to a small pericentric distance. They can also collide with or become captured by the perturber. Moons that are thrown onto hyperbolic orbits can enter heliocentric orbits or get ejected out of the system as free-floating moons. Ejection is by far the most common outcome. Most of the moons that are scattered onto heliocentric orbits experience further close encounters with the giant planets, because the giant planets themselves are on eccentric orbits. In our 100 million year simulations,

94% of the moons on heliocentric orbits become ejected from the system to beyond 1000 AU. Depending on the formation efficiency, free-floating "moons" can outnumber free-floating "planets" (Veras & Raymond, 2012).

Perturbation from planet orbital evolution

Planetary orbits evolve chaotically as they scatter off each other and secularly interact in the planet-planet scattering phase. Since planetary orbits set the moon stability criteria, moons are hosted in an environment constantly evolving with respect to the stability limit. Different mechanisms leading to the destabilization of moons can be switched on besides direct perturbation in planetary close encounters. The following subsections discuss relevant stability criteria.

Hill stability The size of the Hill sphere depends on the planet's semi-major axis, eccentricity, and inclination (Domingos et al., 2006, Donnison, 2010). Planets migrate in/out during the instability phase, or their distance from the star varies within an orbit due to a rise in eccentricity. When their Hill sphere shrinks through this path, moons that become exposed outside of the Hill sphere become destabilized without the direct perturbation of a close encounter. The planet's inclination, coupled with eccentricity, also determines the Hill stability.

Critical semi-major axis The critical semi-major axis of a planet separates the region where the perturbation from planet oblateness and the central star dom-

inates respectively.

$$a_{crit} = \left(2\, J_2\, R_p^2\, a_p^3\, (1 - e_p^2)^3\, \frac{m_p}{m_*} \right)^{\frac{1}{5}}, \tag{3.5}$$

in the co-planar case, where J_2 is the quadrupole moment of the planet, R_p the planet's radius, a_p the planet's distance from the star, m_p the planet's mass, and m_* the star's mass (Deienno et al., 2011, Kinoshita & Nakai, 1991, Nicholson et al., 2008). In this work, moons often have inclinations high enough for a_{crit} to depend also on their inclinations. Well inside a_{crit} the orbital angular momentum axis of the moon precesses rapidly around the planet's spin, and well outside a_{crit} it precesses around the orbital angular momentum axis of the planet. Moons close to a_{crit} precess about local Laplace planes that lie in between the spin and orbital plane of the planet. As a_{crit} of a planet coevolves with its orbit, the center of precession of moons switches between different planes, thus changing the moons' inclinations relative to their local Laplace planes. Some moons wander close to a_{crit} in systems with high obliquities, and could possibly be destabilized through this path (Tremaine et al., 2009), although planets evolving to high obliquities through planet-planet scattering also often acquire high inclinations and eccentricities, making it hard to pin down the exact cause of the moon's orbital instability. Also, in high obliquity systems, the moons' inclination relative to the center of precession could change significantly as a_{crit} changes, giving them stability or destabilizing them. When a planet evolves to high inclinations, a path may be open to destabilizing its moons. As planet inclination reaches $\sim 40^\circ$, the Kozai mechanism starts to act on the moon that then stays close to and precesses around the orbital plane, causing the eccentricity and inclination to oscillate in a coupled manner.

The scenarios above do not involve any direct perturbation from planetary close encounters, but are caused by evolved planetary orbits induced by close encounters plus mutual secular perturbation. As a side note, however, only a small fraction ($< 5\%$) of surviving planets or planets within 100 AU from the central star in planet-planet scattering have inclinations above 40° (Chatterjee et al., 2008, Raymond et al., 2010), so this is not an important mechanism to destabilize the orbits of moons in the planet-planet scattering scenario.

Evolved moon orbit

Besides the evolution of planetary orbits, the evolution of the orbits of moons can also destablize or change themselves by evolving beyond the stability limit. Oftentimes, the orbits of moons evolve when there is an effective close encounter, although mutual interactions among planets when they are outside each other's Hill sphere also affect moons. As their orbits change, they can become exposed outside of the Hill sphere and thus become destabilized or more stable. They can also shift across a_{crit} as they radially migrate in and out, which cause changes in their inclinations relative to the center of orbital precession (local Laplace plane). Close encounters can throw them onto random inclinations relative to their center of precession and affect their stability.

Secular evolution of moon post instability

After the planetary system has been restored to stability via removal of 1 or 2 planets, there are no more planetary close encounters. Most of the moons simply evolve in a stable manner, with their semi-major axis staying fixed, and

eccentricity and inclination undergoing sinusoidal oscillation periodically as a consequence of the precession of their orbital angular momentum around the local Laplace plane. For some moons inside a_{crit} with inclinations greater than 40° from the equatorial plane, the inclination and eccentricity will be slowly oscillating, out of phase with each other by 180°, indicating that the Kozai mechanism is in action. Moons with inclinations under 40° undergo secular evolution, with a fixed semi-major axis and periodically oscillating eccentricities and inclinations. The same argument applies to moons exterior to a_{crit} inclined relative to the orbital plane. Those that undergo Kozai evolution with high inclinations could become highly eccentric or unstable in the longer term.

3.4.3 Statistics

Table 3.1 summerizes possible dynamical outcomes for moons and the respective probability. The moons that remain planet-bound may stay around their original host planet, around a new host planet that captures it away from the original, or around a planet that itself is ejected as a free-floater. Other moons may stay on heliocentric orbits, or collide with the host planet, the perturbing planet, or the star. Multiple close encounters throughout the planet-planet scattering phase are very efficient for removing moons. 17% of the moons in the orbital range of 0.01 – 0.35 R_H remain bound to the host planet after 1 million years of integration. This survival rate with regard to the entire moon population signifies how much of its parameter space allows for moons to be stable. Inferring from the above numbers, and collapsing the parameter space to the innermost region, the moon stability limit is $\sim< 0.1\ R_H$. For a similar argument from a different perspective, of the 17%, 88% are initially within 0.1 R_H. In other

words, planet-planet scattering truncates the primordial moon disk at least as close as 0.1 R_H. If considering only moons interior to 0.04 R_H, inside which the regular moons of the Solar System gas giant planets are located, moon survival rate amounts to $\sim 42\%$, forecasting a decent survival chance for moons on Galilean-moon like orbits. The rest of the destabilized moons (83%) are removed via various paths, as shown in table 3.1. Ejection out of the system is the most common outcome ($\sim 41\%$), given the small mass of moons relative to the planets (here moons are massless). Collision of moons with the other big bodies is also equally dominant($\sim 37\%$); among them, collision with the planets are more frequent than with the central star due to the proximity of moons with planets, especially during planetary close encounters. The above two paths remove the majority of the moons. The remaining unstable moons stay in different places in the system; $\sim 2\%$ of them are captured by the perturbing planet, and $\sim 2\%$ of them orbit planets that gets ejected out of the system as free-floaters. Capture of moons via exhange when two planets approach each other closely is a much less efficient way of producing irregular satellites than capture from circumstellar materials probably because of the large number of the latter and that it requires a particular configuration – small d_{min}. After 1 million yr, $\sim 22\%$ of all moons initially stable around their host planets have destabilized and ended up orbiting the star; however, further integration to 10^8 yr removed most of these heliocentric "moons", with the rate reduced to $\sim 1\%$. The moons removed by 10^8 yr are incorporated into the final statistics in this section.

10^6 years is 1-2 orders of magnitude shorter than required for a subset of simulations to finish the instability phase. A sample subset of simulations taken from those that haven't ended the instability phase are further integrated up to 10^7 yr. The survival rate of moons in the subset decreases in half for moons

Outcome(Surviving)	probability(%)
Host planet bound	17
Captured by perturber	2
Free-floater bound	2
Heliocentric	1
Outcome (removed)	**probability(%)**
ejected	41
Collision with planet	20
Collision with star	17

Table 3.1 Probability of each dynamical outcome for all moons in the 10^6–10^8 yr simulations in set 2.

across all orbital distances. Extrapolating from the subset, the survival rate of the Galilean moons after instability ends can be around 20–40%, lower than predicted by the 10^6 yr simulations in set 2. The number of close encounters and the magnitude of their perturbations between 0–10^6 yr and 10^6–10^7 yr are comparable. Although the temporal length of the latter makes perturbation from close encounters more diffused, the change in survival rate and how perturbed the surviving moons' orbits are not insignificant.

3.4.4 Survival factors

Key factors for moon survival include various system parameters. Parameters not observable in actual systems include the initial orbital distance of moons and properties of planetary close encounters such as closest approach distance, number of close encounters, and close encounter geometry. Observables include planet mass and planetary orbits (semi-major axis, eccentricity, inclination, and obliquity). All of the above parameters work together in determining the dynamical outcome and stability of moons, so how moons' orbits would change can only be predicted with a single parameter in a probabilistic sense. Moreover,

the long length of the simulations and the large number of close encounters (the average number of close encounters in a 10^6 yr simulation is 224), or the large removal rate of moons probably smears out any clear correlation between some of the parameters and the moon survival rate within the simulated set. Figures 3.5 and 3.6 compare various factors with moon survival rate and dynamical outcome. The stability limit of moons around the planet is defined at the location of the outermost surviving moon in the Hill sphere. Moon survival rates are nearly equivalent to the stability boundary because moons are all set at different orbital distances around each planet, and due to strong perturbations by close encounters, survival of an outer moon often ensures that moons interior to it also survive. However, the encounter geometry sometimes adds uncertainties to the relation between moon survival and the stability boundary. Note that parameters related to close encounters do not correlate with moon survival rates in the full-length simulations, unlike what has been shown in single flyby events in section 3.3.

Moon initial semi-major axis

A_i **vs. survival rate:** The planetocentric distance of moons determines their survival rate. Quite intuitively, inner moons are more likely to survive because they have several advantages. As a typical case in figure 3.4, after the perturber passed by, the inner moons usually experience less change in their orbits than do the outer moons. Closer-in moons also have a lower probability of being perturbed by an encounter since moons are not sensitive to encounters far from them. In figure 3.4, in both of the close encounter events, most of the moons interior to the perturbing planet experience little perturbation while those exte-

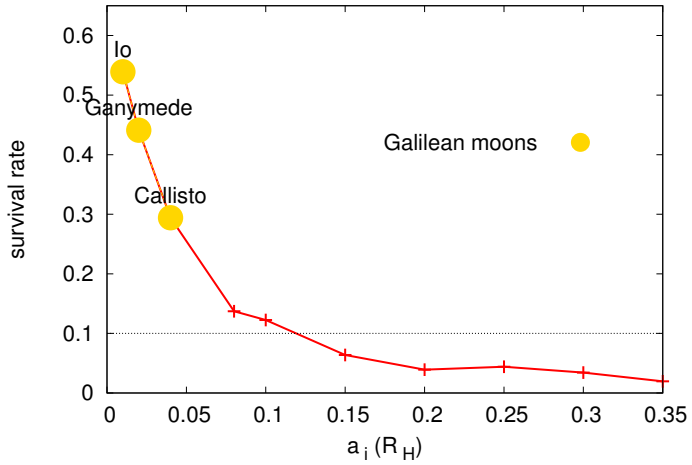

Figure 3.5 Moon initial semi-major axis (a_i (R_H)) vs. moon survival rate. The moon survival rate is predetermined by the moon's initial position in the host planet's Hill sphere. In the semi-major axis range 0.01 - 0.04 R_H, which is roughly where the Galilean moons are located (only three Galilean moon's orbits are sampled in this work), survival rate drops exponentially.

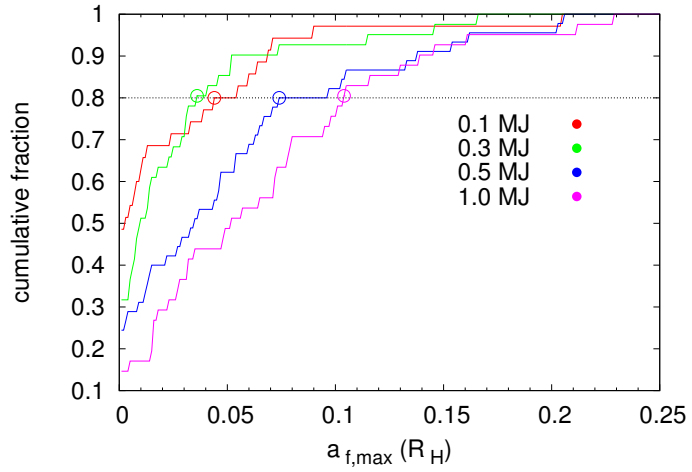

Figure 3.6 Planet mass vs. moon stability limit. The axes of the figure represent the cumulative fraction of planets in semi-major axis of their outermost surviving moon (in R_H). Different colors represent planets with different masses in M_J.

rior to and near it become destabilized or highly eccentric. Different encounter geometries do add some randomness to the trend.

As shown in figure 3.5, survival rate drops rapidly as a moon's semi-major axis increases. Moons at Io-like distance (0.01 R_H) have a survival probability $\sim$0.5–0.6. Beyond 0.1 Hill radii, the chances of survival are very low (< 0.1). The survival rate for the Galilean satellites by the end of 10^6 yr could be higher than ~ 0.3–0.5 as predicted by the simulations (the outeremost satellite, Callisto , is at $\sim 0.04\ R_H$ from Jupiter), because in the simulations they orbit around planets of all masses but in the dynamical history of the Solar System Jupiter was the most massive planet by the time the satellites formed (Canup & Ward, 2009). Their survival rate could also be lower if the Solar System hasn't ended its instability phase by 10^6 yr.

A_i vs. dynamical outcome: The dynamical outcomes of moons are predetermined in a probabilistic sense by their initial location within the Hill sphere. Figure 3.7 shows the cumulative distribution in the initial semi-major axis of moons that have different dynamical outcomes. The overall population (black line) divides the planet-bound moons and the unbound moons. Not surprisingly, the planet-bound moons tend to occupy the inner part of the Hill sphere, with their curves consistently lying on the left divison of the overall population. Among them, those that stay stable around an ejected free-floating host planet lie in the most interior region, likely due to the fact that they need a sufficiently strong gravitational binding and enough separation with the perturber to survive the planet-ejecting close encounters. By contrast, moons that become heliocentric, including those that later collide with or orbit the star and those that are ejected out of the system, lie consistently on the right hand side of the overall population, and the three curves are very close to each other, since right after the close encounter that destabilize them, they share the same outcome.

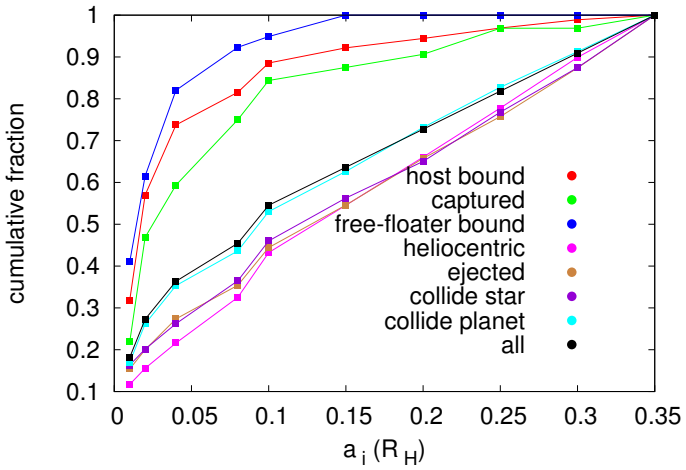

Figure 3.7 Moon initial semi-major axis a_i (R_H) vs. moon dynamical outcome. Where moons will end up in planetary close encounters are also predetermined by their initial semi-major axis, not definitely but probabilistically. In the color legend from top to bottom are moons (1) bound to the original host planet (red), (2) captured by the perturber (green), (3) bound to the free-floating planet (blue), (4) orbiting the star (pink), (5) ejected out of the system as free-floating moons (brown), (6) colliding with the star (purple), (7) colliding with planets (light blue) , followed by the overall moon population in black.

Planet mass

The mass of the host planet determines the extent of its moons' gravitational binding, which if large compared to perturbations, the moons are more likely to survive. Therefore the more massive the planet, the more stable its moons usually are. Figure 3.6 shows the cumulative distribution of the outermost surviving moon in semi-major axis on planets with different masses. Moons around more massive planets have wider distributions across the Hill sphere; in other words, they are more likely to have a larger stability boundary. To have moons surviving beyond 0.1 R_H, the probability for planets with M_J, 0.5 M_J, 0.3 M_J, and 0.1 M_J are $\sim 35\%$, $\sim 20\%$, $\sim 10\%$, and $\sim 5\%$ respectively. Around 0.5 and 1 M_J planets, the 80th percentile of the outermost stable moons reaches 0.1 R_H,

whereas around 0.1 and 0.3 M_J planets they are under a much tighter limit of 0.04 R_H. All planet masses from 0.1 to 1 M_J permit moons on Galilean satellite orbits to have good survival rates.

3.4.5 Final orbits of moons

Moons with different dynamical outcomes have experienced different amount of perturbations, so they occupy different parts of the orbital parameter space; in other words, they are dynamically distinguishable. As a general trend, the more they deviate from their initial orbits, the more they have been perturbed.

Δ a

Planet-bound moons have low tolerance for radial migration. A large amount of change in the moons' semi-major axis often destabilizes it. Moons that stay bound to stable host planets experience no dramatic changes in semi-major axis. 80% of them migrated outward / inward by less than 46% / 32% in semi-major axis (or %90 by less than 73%/45%). Moons stably orbiting ejected free-floating planets (2%) are the most tightly clustered at small semi-major axis (in AU). The 80th percentile reaches as far as $\sim$ 0.01 AU. Their small orbital distance is very likely a result of the same processes identified in the previous section. In comparison, moons that have ended up on heliocentric orbits have a wider distribution in semi-major axis than the surviving planets.

Eccentricity

Quite intuitively, moons that stay stable around their original host planet have less excited orbits than those that are destabilized. Moons that remain stable around bound or free-floating planets have lower average eccentricities, and those captured by another planet or go into heliocentric orbits are much more eccentric.

Figure 3.8 shows the cumulative fraction in eccentricity of moons of different dynamical outcomes. Primordial moons bound to the host planets are, to no surprise, the least eccentric population, since their orbits receive the least perturbation and change the least. (fig. 3.8) 44% of them have eccentricities under 0.01, and the average eccentricity of the perturbed ones is 0.31. Moons become eccentric once an encounter strong enough to perturb them occurs, and the probability to recover via subsequent encounters is extremely low; more violent encounters usually only increase eccentricity by a large amount, and milder encounters have a chance to damp down the eccentricity but only by a moderate amount. The average eccentricity of moons bound to free-floating planets is $\sim$ 0.32, higher than moons around stable planets, and the 80th percentile reaches e $\sim$ 0.6; as a comparison, the 80th percentile of primordial moons around stable planets reaches e $\sim$ 0.47. Moons that are captured away by a different planet have a higher average eccentricity of 0.57, and they are more evenly distributed across the eccentricity spectrum except at the low end with e $<$ 0.2, because their orbits are randomized by the capture process.

Moons stripped away from the planet to orbit the central star have the highest average eccentricity of 0.68, the 80th percentile reaches as high as e = $\sim$ 0.91, much higher than all the planet-bound populations, due to the higher toler-

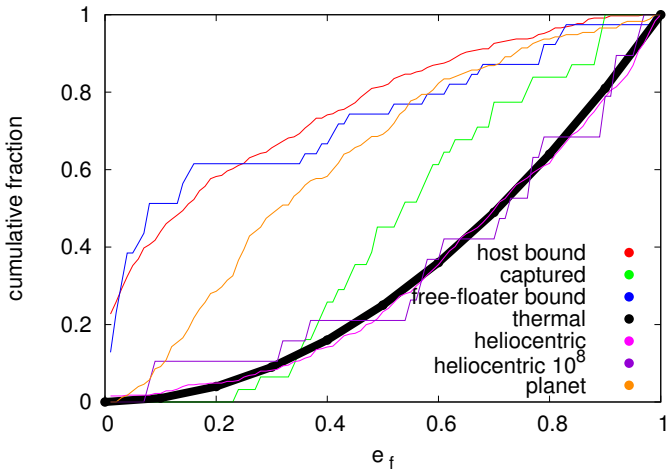

Figure 3.8 Cumulative fraction of final eccentricity of stable planets and moons with different dynamical outcomes. In the legend different colors represent different poulations of moons. For planet–bound moons, those bound to the original host planet are represented in red, those captured by the perturber in green, and those bound to free-floaters in blue. Moons on heliocentric orbits by the end of $10^6/10^8$ years are represented in pink / purple. Planets surviving around the central star are represented in black.

ance for eccentricity in the heliocentric system than in the planetocentric system. Both for simulations lasting 10^6 and 10^8 yr, all the heliocentric moons in the set form a thermal eccentricity distribution (fig. 3.8),

$$f(e) = 2e. \tag{3.6}$$

This distribution resembles that of wide binary stars and suggests that they have been sufficiently pulled and kicked around for the energy to approximate an equilibrium state (Jeans, 1919, Kouwenhoven et al., 2010). However, as a consequence of extremely high eccentricities and the fact that the planets are also on eccentric orbits, interactions are often strong and frequent, making heliocentric moons extremely vulnerable to subsequent removal. By the end of 10^8 years, only $\sim 6\%$ of them remain in the system ($a < 1000AU$). However, if a planetesi-

mal belt exists at the edge of the planetary system, it could help trap moons on the way to ejection, turning them into KBO-like objects (Raymond et al., 2009). Another possibility to produce KBO-like objects out of ejected moons is that, if some instabilities happen before the dissipation of the gas disk, gas drag could act to trap a heliocentric moon by lifting its pericenter from the grasp of the perturbing planets (Raymond & Izidoro, 2017). Of the majority of moons venturing beyond 1000 AU (classified as "ejected" in this work), $\sim 35\%$ are able to reach at least 5000 AU with eccentricities $\sim< 1$ within 10^8 yr. Those moons might have a chance to damp down their eccentricities by the galactic tide and become Oort-cloud-like objects (Morbidelli et al., 2005, Tremaine, 1993). 20% of the test simulations for Oort-cloud like candidates with fractional angular momentum change larger than 10^{-5} are ruled out because most of the moons that reach the orbital distance where the galactic tides becomes effective have eccentricities $\sim< 10^{-5}$ below 1.

Inclination

Like eccentricity, inclination is an indicator of the dynamical history of moons. Figure 3.9 shows the cumulative fraction of moons in inclination of different dynamical outcomes. The stable populations, the host-planet-bound and free-floater-bound moons and the planets are again the least evolved in i, as in a and e. The more unstable moon populations such as the heliocentric moons and the captured ones have larger inclinations; 45% of them are above 40° as seen from the planet's orbital plane. And a much larger fraction of them are retrograde than other populations; 23% for moons captured by the perturbing planet, and 16% for heliocentric moons by 10^6 yr. In comparison, only $\sim 1\%$ of

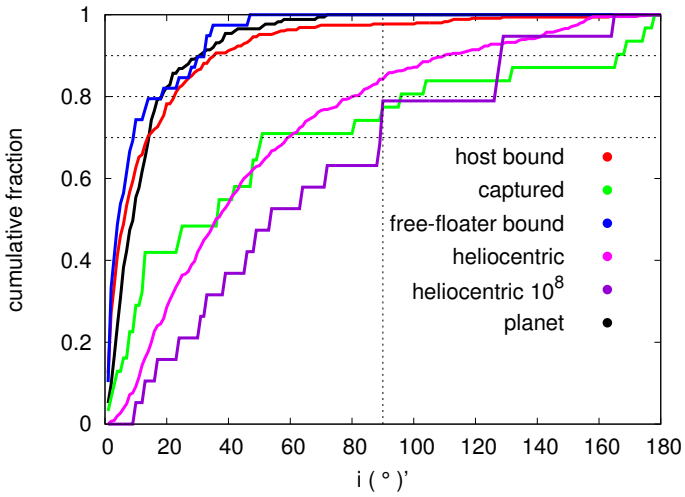

Figure 3.9 Cumulative distribution of final inclination of stable planets and moons with different dynamical outcomes. The figure legend is identical to fig. 3.8. Inclinations are measured relative to the initial zero inclination plane of the system, if not specified.

the planet–bound moons are on retrograde orbits.

Surviving moons have relatively quiet orbits, although still significantly more perturbed than the Solar System regular moons. 86% of stable primordial moons within a_{crit} are under $\sim 20^\circ$ as seen from the host planet's equatorial plane and 73% are above 1°, as for those exterior to a_{crit}, 81% are under 20° from the host planet's orbital plane. The inner population is significantly less perturbed than the outer one. As for moons stable around ejected free-floating planets, 80% are under 20° from the initial reference plane, but the number of such systems is small, making statistical interpretation difficult. Both of the primordial populations have under 2% on retrograde orbits.

Moons captured by the perturber are more evenly distributed across the inclination spectrum, including a significant portion on the retrograde side, and in general they have higher inclinations than moons bound to their original host planet; this tendency is similar to the simulated capture of satellites from plan-

etesimals (Nesvorný et al., 2007).

3.5 Occurrence of free floating exomoons

Here we calculate the abundance of free-floating exomoons and estimate its potential contribution to the vast number of free-floating objects estimated from microlensing observations, since planet-planet scattering appears to be very efficient at ejecting moons. Sumi et al. (2011) obtained an observed frequency of free-floating planets per main-sequence star $\frac{N_{ff}}{N_{star}} = 1.8^{+1.7}_{-0.8}$, and Mróz et al. (2017) found $\frac{N_{ff}}{N_{star}} = 0.25$ for Jupiter-mass free-floaters. The equation below adopts a smilar method to Veras & Raymond (2012) to estimate the number of free-floating moons:

$$\frac{N_{moon,ff}}{N_{star}} = f_{gp} \times f_{system,unstable} \times n_{gp,eject} \times n_{moon,eject} \tag{3.7}$$

, where N_{star} is the total number of stars, f_{gp} the fraction of stars with giant planets, $f_{system,unstable}$ the fraction of planetary systems that go unstable, $n_{gp,eject}$ the averaged number of ejected giant planets per system, n_{moon} the averaged number of moons per system.

The giant planet occurrence rate f_{gp} has been measured by radial velocity surveys to be $\sim$0.1-0.2 for Sun-like stars (Cumming et al., 2008, Mayor et al., 2011, Rowan et al., 2016) but with a stellar-mass dependence such that low-mass stars are deficient in gas giants (Johnson et al., 2007, Lovis & Mayor, 2007, Winn & Fabrycky, 2015, Wittenmyer et al., 2016). Given the predominance of low-mass stars by number, we expect the average f_{gp} to be in the 1–10% range. The

number of giant planets that participate in a given instability $n_{gp,unstable}$, must be at least two, and may be larger. We assume an average of 2–4. To match the observed eccentricity distribution of exoplanets, the fraction of unstable planetary systems is very high. $f_{system,unstable}$ is at least 75% but is more likely more than 90% (Raymond et al., 2010, 2011). $n_{moon,eject}$ equals the number of moons per system $\times$ ejection rate. The former is unknown, and we assume a value of 1–5; the latter is roughly 40% from the result of simulation set 2. Therefore, the occurence of moons per star is $\mathcal{O}(0.01\text{–}1)$, which predicts a galactic population of free-floating (former) moons that may be as abundant as stars. WFIRST can detect such objects down to ~ 0.1 Earth masses (Spergel et al., 2015).

Another interesting population of moons that sit stably on ejected planets has an occurence probability of order 1% per system based on the theoretical result in this work. Replacing $n_{moon,eject}$ with the fraction of free-floating giant planets that carry moons (~ 0.08) and using the same estimate for other parameters as above yields an occurence rate of $\mathcal{O}(10^{-3}\text{–}10^{-2})$ per star. If using the observed frequency of free-floating planets per star (0.25) (Mróz et al., 2017) and multiplying it with the fraction of moon-bearing free-floating planets from the simulations (~ 0.08), the occurence of moon-bearing free-floating planets per star is $\mathcal{O}(10^{-2})$. Both approaches yields probabilities that are not insignificant.

3.6 Conclusion

We have directly simulated the survival of exomoons during giant planet scattering. To summerize, most moons are unstable during planet-planet scattering and they have rich dynamical outcomes. Planet-planet scattering is a very ef-

ficient way to destabilizing moons, mainly because the close approach of the relatively massive perturbing planet compared to moons, and secondly because planets themselves also experience strong mutual perturbations induced by the instability even outside of close encounters. Only $\sim 10-20\%$ of the moons within 0.35 R_H remain bound to the original host planet; in other words, only $\sim 10-20\%$ of the orbital parameter space in the Hill sphere allows for moons to be stable. Close-in moons on Galilean-moon like orbits (0.01 - 0.04 R_H) have at least twice higher survival rates than the whole population. The majority of moons ($\sim 80-90\%$) collide with big bodies or become ejected out of the system. Given the high probability of ejection, there is likely a population of free-floating objects in interstellar space that were born as moons of giant planets, with an occurrence rate of $\mathcal{O}(0.01\text{–}1)$ per star. Future microlensing detection is capable of observing such objects down to ~ 0.1 Earth masses. A very tiny fraction of moons live on interesting orbits, such as around free-floating planets (occurrence rate $\mathcal{O}(10^{-3}\text{–}10^{-2})$ per star), around a perturbing planet as a captured object, or on heliocentric orbits. As moons around free-floating planets tend to be very close-in and eccentric in the planet-planet scattering scenario, there may be a good chance of having tidally heated Io's around free-floating planets. The simulated number of moons are not abundant enough to explain the efficiency of producing Trojan-like objects from moon scattering, but intuition and the simulation results for moons on heliocentric orbits suggest that such objects can have a hard time surviving the instability phase. Since planet-planet scattering tends to destroy already-formed terrestrial planets (Carrera et al., 2016, Matsumura et al., 2013, Veras & Armitage, 2005) or their building blocks (Raymond et al., 2011, 2012), a possible way to form terrestrial planets could be from the heliocentric moons remaining in the system after planet-planet scattering,

although the simulation results do not predict it as a highly efficient mechanism (probability ~ 0.01).

Properties of the close encounters, planets, and moons all come into play together in determining the stability and dynamical outcomes of moons. Among all factors, the planet mass and the initial orbits of moons stand out as clear predictors of a moon's final outcome, though only in a probabilistic sense. The source region of moons with different dynamical outcomes can be anticipated to some degree.

As to the final orbits of moons, stable moons experience limited radial migration, so it is not a common phenomenon for moons to swap their orbits. However, moons do move in and out of the planet's Hill stability limit and the critical semi-major axis during the instability. For the Solar System gas giant planets, moons interior to the critical semi-major axis are co-planar with the equator and have tiny eccentricities (Deienno et al., 2011), whereas moons exterior to it are inclined and eccentric Jewitt & Haghighipour (2007). In contrast, in the bulk simulated extra-solar systems, moons interior to the critical semi-major axis have a tiny probability $\sim< 0.1$ of having unperturbed orbits like in the Solar System, giving a hint that Solar System gas giant planets might not have experienced planetary close encounters, or that they experienced very few and very mild close encounters. In the region exterior to the critical semi-major axis and interior to the Hill stability limit for prograde moons, moons have a negligible probability ~ 0.01 of having unperturbed orbits, just like in the Solar System. The Solar System irregular moons on prograde orbits could have mixed origins of primordial populations and external capture from circumplanetary materials. However, tests in the planet-planet scattering scenario show that moons on

retrograde orbits are not produced very efficiently, therefore supporting the origin of retrograde irregular satellites via other mechanisms such as capture from circumplanetary materials (Nesvorný et al., 2007).

CHAPTER 4

TILTING EXTRASOLAR GIANT PLANETS IN PLANET-PLANET SCATTERING

4.1 Introduction

Extrasolar planets have very different properties than our Solar System planets. For example, since the early era of discovery, extrasolar planets have been found to be much more eccentric. Among models of extrasolar planet formation, planet-planet scattering best reproduces the above mentioned property (Chatterjee et al., 2008, Raymond et al., 2010). Besides producing highly eccentric planets, planet-planet scattering also produces planets on highly inclined orbits. Since planet-planet scattering produce dynamically excited systems, through strong mutual planet-planet interactions, grazing planetary close encounters (down to the order of 0.001 AU), and planet migration, planet-planet scattering is likely to provide a path to tilting planets to high obliquities. In this work, we test planet formation models with planet obliquity, in the dynamical setting of planet-planet scattering. To date, planet obliquity (defined in this work as the angle between the planet's spin aixs and orbital normal) has not been observed. Throughout this work, planetary close encounters are defined as when planets come into each other's Hill sphere.

4.2 Numerical method

This numerical scheme of this project adopts the same tool and approach as in chapter 3, as the numerical scheme is designed for close planet-planet interactions and planet spin evolution. Planets are treated as oblate. A non-secular equation of motion for spin evolution is applied, since the timescale of dynamical evolution in planet-planet scattering is short, especially during planetary close encounters. The integration algorithm applied is the conservative Bulirsch-Stoer algorithm. The oblateness of planets and planet spin evolution are included in the integration. Planets are treated as rigid bodies. Change of planet spin from tidal distortion during planetary close encounters is neglected since the timescale of planetary close encounters are typically short, from the order of a few days up to 100 years. The initial integration time step is 1 day per step. The timestep automatically adjusts itself to maximize integration efficiency as long as the integration error falls under the error tolerance. The planet oblateness is set as identical to that of Jupiter (J_2 = 0.0147) for all planets, as the value fits the equation of state for our Solar System gas giants. The rotation period Ω of all planets is set to be 9 hours, identical to that of Jupiter, and it stays constant through out the simulation (eq. 4.1). The spin components of the planets are unit vectors that change directions with fixed magnitudes. The maximum allowed deviation of orbital energy from perfect energy conservation (dE/E) is 10^{-4}.

4.3 Simulation Setup

The simulations in this work consist of one base simulation set and six other comparison simulation set. Each of the comparison set has one or two variables in the initial setting different from the base set (Fig 4.1). The base simulation set (set 1) contains 68 simulations, each of which are three planet systems. Each simulation in the base set is configured with a Sun like central star and three giant planets. The planets' masses are sampled from four different planet masses: 0.1 M_J, 0.3 M_J, 0.5 M_J, 1 M_J. There's one simulation for each unique arrangement of the masses of the three planets in the simulation set, but systems with equal mass planets have two simulations for each configuration. The giant planets are piled up in between 5 and 10 AU. Their initial mutual spacing are 3.5-4.5 mutual Hill radii. The separation is set so that planetary close encounters can be triggered within a reasonably short time to allow for the simulations to finish their dynamical instability within the simulation time of 100 million years. Planets have initial spin angle $= 0.1 - 1^\circ$ from the initial zero inclination plane (z-axis), so they have a small initial tilt from their orbital axis, to facilitate a natural evolution of spin. The planets have initial eccentricities = 0.02-0.1 and inclinations $= 0, 0.01, 0.02^\circ$ from the initial zero inclination plane (z axis). The eccentricities facilitate planet-planet scattering to happen within a reasonable amount of time, and the inclinations avoid immediate collisions between planets right at the start of the simulation. The initial orbits satisfy the natural outcome of planet formation (Chatterjee et al., 2008). The orientations of the spin and orbital axes are randomized. The simulation duration is set at 100 million years, in reference of literatures that simulate planet-planet scattering (Chatterjee et al., 2008, Jurić & Tremaine, 2008, Raymond et al., 2010). Within 100 million years,

almost all simulations can finish their planet-planet scattering phase, with one or two planets removed from the system via collision with the star or ejection from the system. Systems with planetary collisions are ruled out from the final statistics because the simple treatment of planet merger of transferring orbital momentum to spin momentum in N-body simulations is not appropriate for accurate planet spin evolution. Realistic treatment with fluid dynamics is further required. The comparison simulation sets copy the base set and change one or two variables for the simulations (Fig 4.1), in order to see whether a change in planet-planet interaction will affect the obliquity of planets. Simulation set 3 shifted the inner most planet closer in to 3 or 1 AU, in order to explore whether a change in planet-planet interaction due to a different Safronov number can affect the obliquity evolution of planets differently. Simulation set 4 reduces the smallest planet mass in half and increases the largest planet mass by twice, so that the pool of planet masses to be sampled in each simulation is 0.05 M_J, 0.3 M_J, 0.5 M_J, 2 M_J. In simulation set 5, torques from other planets are not included in the spin equation of motion. Only stellar torques affect the spin evolution of planets, in order to study whether the contribution of other planets' torques are significant for planet spin evolution. In simulation set 6, the mass of the central star is reduced to 0.5 Solar masses. set 7 only contains systems with three equal mass planets. In set 7, each of 32 simulations use three M_J planets each, each of the 32 simulations use three 0.3 M_J planets , and each of 30 simulations use three 0.1 M_J planets.

Initial setup : base simulation set 1

body	mass
star	1 solar mass
planet 0	0.1/ 0.3 / 0.5/ 1 Jupiter mass
planet 1	0.1/ 0.3 / 0.5/ 1 Jupiter mass
planet 2	0.1/ 0.3 / 0.5/ 1 Jupiter mass

orbits	
planet 0 - planet 1 separation (mutual Hill radii)	3.5-4.5
planet 1 - planet 2 separation (mutual Hill radii)	3.5-4.5
planet 0: a (au),e,i (degree)	5.0, 0.02 - 0.1, 0 / 0.01 / 0.02
planet 1: a (au),e,i (degree)	x , 0.02 - 0.1, 0 / 0.01 / 0.02
planet 2: a (au),e,i (degree)	~ 7-10, 0.02 - 0.1, 0 / 0.01 / 0.02
planet 0: g,n,l (degree)	0 - 360
planet 1: g,n,l (degree)	0 - 360
planet 2: g,n,l (degree)	0 - 360
planet spin i,n (degree)	0.1 - 1, 0 - 360

Figure 4.1 Initial setup of the base simulation (set 1).

sim set	Difference from base set
1	base
2	PLANET 0 at 3 AU
3	PLANET 0 at 1AU
4	Planet masses 0.05 / 0.3 / 0.5 / 2 M_J
5	no planet-planet torque
6	star: m = 0.5 Solar mass PLANET 0: a= 3AU
7	Equal mass planets

Figure 4.2 Initial setup: simulation set 1-7.

4.4 Result

4.4.1 Spin dynamics of planets

Physical mechanism of spin evolution

The spin evolution of planets are under the influence of external torques provided by other bodies in the system (ie. the star, other planets, planet moons). The spin evolution is governed by the following non-secular equation of motion:

$$\frac{d\hat{s}}{dt} = \frac{-3\,G\,M}{r^3}\frac{J_2}{\lambda\,\Omega}\left(\hat{r}\cdot\hat{s}\right)\left(\hat{r}\times\hat{s}\right), \tag{4.1}$$

where λ is the normalized moment of inertia. λ is taken to be 0.25, close to that of Jupiter, in all simulations. The above equation is derived assuming an instantaneous force and torque within which no variables are averaged over time. It is therefore suitable for the planet-planet scattering case, where planets sometimes interact extremely closely $(ie. \sim 0.001AU)$ over a brief period of time $(ie. \sim 1yr)$. We will look first at spin dynamics in secular systems. In the next subsection, we will derive its application to non-secular systems for a physical insight of planet obliquity evolution in planet-planet scattering. For secular systems with external torques perturbing the spin and orbital angular momenta, the obliquity evolution can be generalized to three different cases. When $\frac{d\hat{s}}{dt}$ (rate of change of spin momentum) is much greater than $\frac{d\hat{L}}{dt}$ (rate of change of orbital angular momentum), the spin axis of the planet will follow its own orbital axis, retaining a constant obliquity (eg. Earth). In cases where $\frac{d\hat{s}}{dt}$ is much smaller than

$\frac{d\hat{L}}{dt}$, the spin doesn't follow the orbital axis. Instead, it statically precesses around the centre of the most dominant torque sources in the system. In a one-planet system, such center is the orbital precession center of the star-planet binary. In this case, the spin axis doesn't get tilted around or evolve. In cases where $\frac{d\hat{s}}{dt} \sim \frac{d\hat{L}}{dt}$, resonance crossing leads to chaotic spin evolution, and thus can generate high obliquity planets (Anderson et al., 2016, Storch et al., 2014, Storch & Lai, 2015). As a side note, the eccentricity evolution during planetary close encounters can alter the rate of spin evolution and thus can contribute to resonance crossings too (Pu & Lai, 2018, Storch et al., 2014).

Comparison between theory and simulation result

This section applies the above theory on a $\sim$ 0.9 million year simulation and gives an in-depth look of how planets gain obliquity during planet-planet scattering. In the following sections, spin evolution is represented by the angle between the final direction of spin and its initial position at t = 0 yr. In the simulation, one planet gained significant spin evolution ($>$ 40 $^\circ$), with two other planet's spin remain largely fixed. The simulation contains an M_J planet at 5 AU (planet 0), and two more distant planets 0.5 M_J at 6.8 AU (planet 1) and M_J at 9.7 AU (planet 2) at t = 0 yr. As shown in figure 4.3, the spin axis of the innermost planet (planet 0) experienced slow and irregular variations, and reached above 50° from its initial alignment. Due to a large variation in its spin axis, planet 0 gained a maximum obliquity of $\sim 55^\circ$. In comparison, the spin of two other planets experienced very little change throughout 1 million years. Figure 4.5 shows the time evolution of obliquity of the three planets. All three planets experience constant large variations of obliquity and planet 0 and planet 1 both

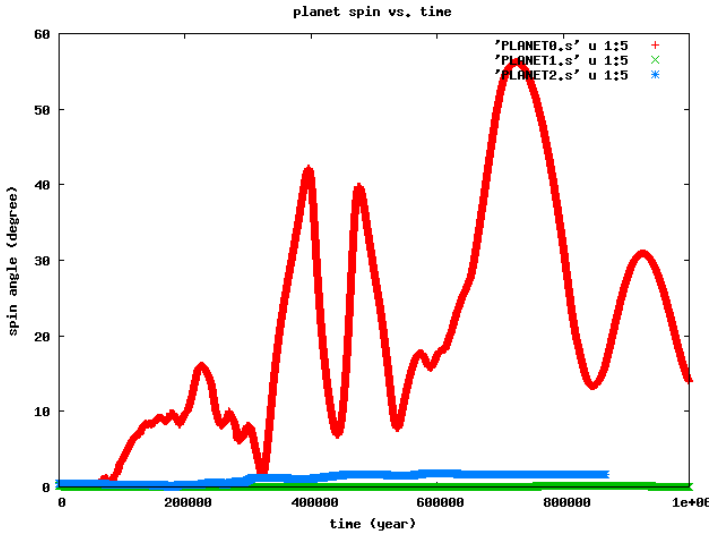

Figure 4.3 Spin evolution of a high obliquity system. Colored lines represent the time evolution of planet spin. Red, green, and blue lines represent planet 0, 1, and 2. Spin angle is defined as the angle between the final direction of spin and the initial direction of spin. Planet 0 has significant spin evolution, which contributes to its high obliquity(Fig. 4.5).

gained high obliquities $> 40°$. A look at figure 4.4 tells is that the dramatic obliquity evolution of planet 0 came from both the contribution of its large variation of spin and orbital inclination. In comparison, the obliquity of planet 1 and 2 is purely from the orbital inclination, since obliquity is defined as the angle between the planet's own spin and orbital plane, and the spin axes of planet 1 and 2 remained precessing close to their initial center of precession.

To investigate what drives the spin evolution of planet 0 and what prevents the spin evolution of planet 1 and 2, , we looked at the time evolution of the ratio of $\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$ for all three planets. As shown in figure 4.6, there are frequent correspondences between the value of $\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$ (log10($\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$) ~ 0), for planet 0. This verifies the theory above, which predicts that resonance crossing can cause complex behaviour of the spin. More intuitively, as both the spin and orbital axis are evolving, a rough correspondence between $\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$ can make

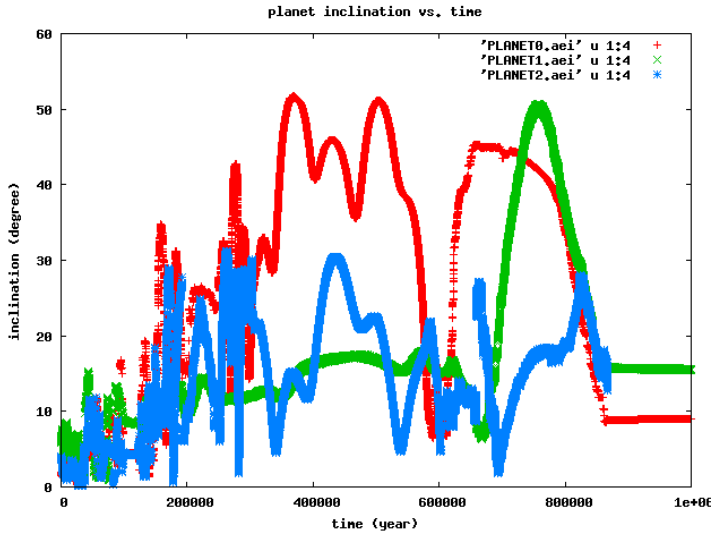

Figure 4.4 Inclination evolution of a high obliquity system. Colored lines represent the time evolution of planet spin. Red, green, and blue lines represent planet 0, 1, and 2. Planet 1 and 2 have significant spin evolution, which contributes to their high obliquity(Fig. 4.5).

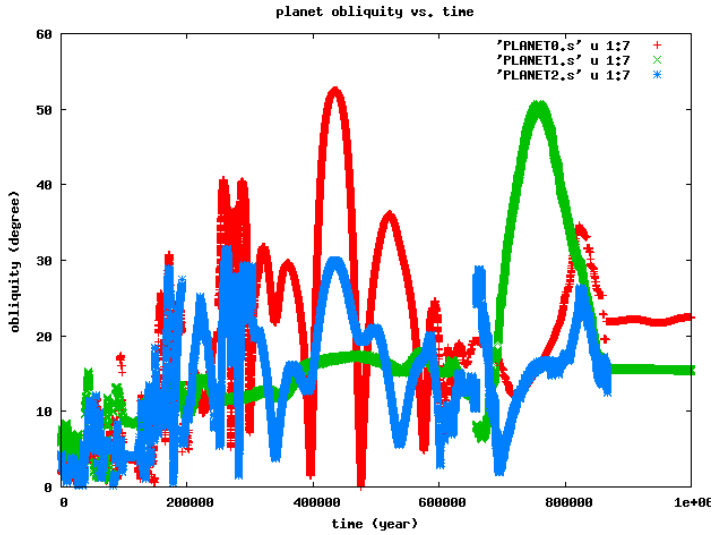

Figure 4.5 Obliquity evolution of planets in a high obliquity system. Colored lines represent the time evolution of planet spin. Red, green, and blue lines represent planet 0, 1, and 2. All three planets gained significant obliquities. Planet 0 and 1 gained maximum obliquities higher than 40°.

the spin feel "lost" . It wouldn't be able to trace the movement of L and precess around it anymore, so the spin evolves. In comparison, in figure 4.7 and 4.8,

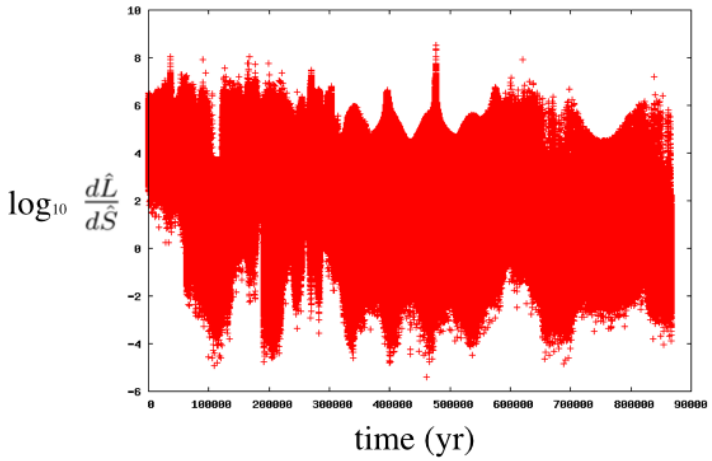

Figure 4.6 $\frac{d\hat{L}}{d\hat{S}}$ vs. time of planet 0.

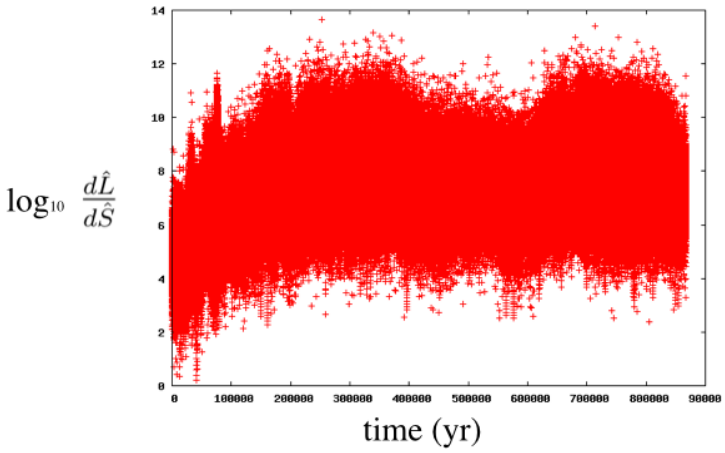

Figure 4.7 Time evolution of $\frac{d\hat{L}}{d\hat{S}}$ of planet 1. L and S are in unit vectors. $\frac{d\hat{L}}{d\hat{S}}$ is plotted in log scale on the y axis.

$\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$ are seldom or never equal for planets 1 and 2. log10($\frac{d\hat{L}}{dt}$ and $\frac{d\hat{s}}{dt}$) are constantly greater than 0. This verifies the theory in the previous section, which predicts that when $\frac{d\hat{s}}{dt}$ is much smaller than $\frac{d\hat{L}}{dt}$, the spin doesn't evolve but stays around a fixed center of precession. Indeed, the spin angle of planet 1 and 2 almost stay fixed throughout the simulation.

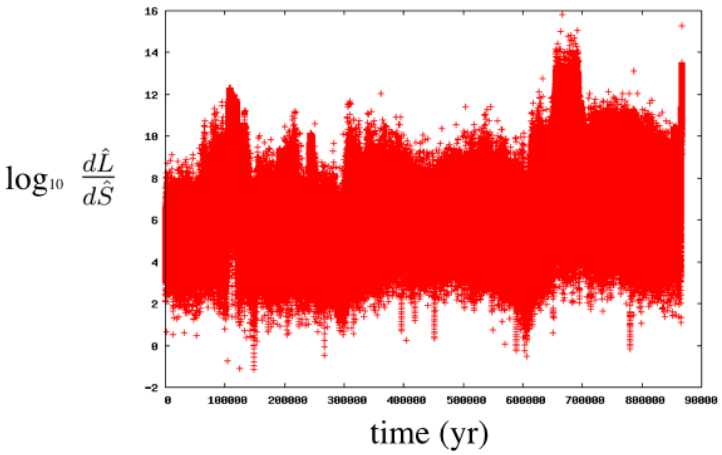

Figure 4.8 Time evolution of $\frac{d\hat{L}}{d\hat{S}}$ of planet 2. L and S are in unit vectors. $\frac{d\hat{L}}{d\hat{S}}$ is plotted in log scale on the y axis.

Torque contribution from the star and other planets

To find out what the main source of torque is that drives the spin evolution of planet 0 in the simulation discussed in the previous sub-section, we compare the torque contribution from the star and planet 2. When planets are well separated, the torque from other planets are negligible, since the torque is proportional to perturber mass. The main contributor of torque is the star. However, during planetary close encounters, the torque contribution from planet 2 can be enormous. At $\sim$ 191620 yr, during the close encounter between planet 0 and planet 2 (minimum encounter distance = 0.008 AU), the torque contribution of planet 2 is 10 million times greater than that of star. And at $\sim$ 148850 yr, during the close encounter between planet 0 and planet 2 (minimum encounter distance = 0.04 AU), the torque contribution of planet 2 is 0.25 million times greater than that of star. However, since the duration of close encounter is typically short, the overall contribution of planet 2 compared to the star is negligible. Over 1 million years, the summed torque contribution from the star on planet 0 is ~ 5660 times greater than that from planet 2. Therefore, the conclusion is the main driver of

planet spin evolution for planet-planet scattering is the stellar torque.

4.4.2 Planetary orbits in planet-planet scattering

This section discusses various orbital and physical properties of the planets after planet-planet scattering, including semi-major axis, eccentricity, inclination, spin angle, obliquity, the number of close encounters experienced. Figure 4.9 shows the final orbital properties of planets in simulation set 1 - 10. The most distinctively different simulation set is set 7, where there're all equal mass planetary systems. The final average orbits of planets appear to be much more perturbed than simulation set 1-9, which have evenly mixed planet mass configurations. In set 7, on average, the surviving planets that still orbit the star stays farther away, with an average $a_f = 31AU$, where as simulation set 1-9 are all under an average of $a_f = 19AU$. The average eccentricity of planets in set 7 = 0.5, also the highest among all. The average planet inclination in set 7 = 26°, topping all other simulation sets. Moreover, planets in simulation set 7 have very high average spin angle and obliquity $> 40°$. The highly perturbed orbits of planets in set 7 comes from the fact that it is harder for them to eject each other. As a result, they interact much more before the dynamical instability ends. As proven in figure 4.9 , each planet in set 7 experienced a much higher average number of close encounters (~ 5500) than other simulation sets ($\sim 400 - 2000$). Due to robust close planet-planet interactions, the planetary collision rate of set 7 is also the highest (50% of the simulations). Figure 4.10 further shows the binned distribution of the counts of planets against the number of close encounters experienced. Simulation set 1 -10 have close resemblance in the shape of the distributions, but compared to sets 4-9, the center of the peak of set 7 is much

more to the right, indicating again the planets experienced many more close encounters in set 7.

As for simulation 4 - 9, except for the final semi-major axes, which are partly affected by their initial starting positions, the final average orbits are quite similar. Planet-planet scattering consistently produce highly eccentric planets, as indicated in the literature. Besides, planet-planet scattering is also able to tilt the spin of planets and produce high obliquity planets. Figure 4.11 is a cumulative plot of planet obliquity. The cumulative plot tells what fraction of planets are under a certain obliquity. The figure shows that for simulation set 1 - 9 , there're about 80-90$\%$ of planets with obliquities $< 40^{\circ}$. In other words, 10-20$\%$ of planets gained obliquities higher than 40°. set 7 significantly deviates from all other simulation sets, with $\sim 35 - 40\%$ of planets with high obliquities. Figure 4.12 shows the cumulative distribution of planets with respect to their final spin orientation from the initial spin angle. It has very good similarity with figure 4.11, which means that the obliquities of planets are not from the evolution of orbital inclinations alone. Planet-planet scattering indeed is quite capable of tilting the spin of planets and can produce some high obliquity planets.

4.4.3 Correlations between orbital and physical parameters

This section discusses correlations between the final orbital and physical properties of planets, and the physical mechanism that links up the correlations. Figures 4.13 and 4.14 show correlations between the planets' final $a, e,$ and i after planet-planet scattering. In figure 4.13, a clear correlation between a and e for a $<$ 5 AU can be observed, as well as for a $>\sim$ 10 AU. The farther the

sim set	a_f (au)	e_f	i_f (°)	s_f (°)	o_f (°)	# enc per pl	collision rate
4	19	0.4	11	13	13	1097	0.14
5	9	0.3	10	16	16	1166	0.18
6	3	0.4	12	20	16	2133	0.37
7	10	0.3	8	12	13	997	0.16
8	13	0.4	12	16	16	1673	0.22
9	6	0.3	8	13	12	397	0.47
10	31	0.5	26	50	45	5515	0.48

Figure 4.9 Average final orbit, encounters, and collision rate of all simulation sets. In the column labels (first row), s_f represents the final spin angle, and s_f represents the final obliquity.

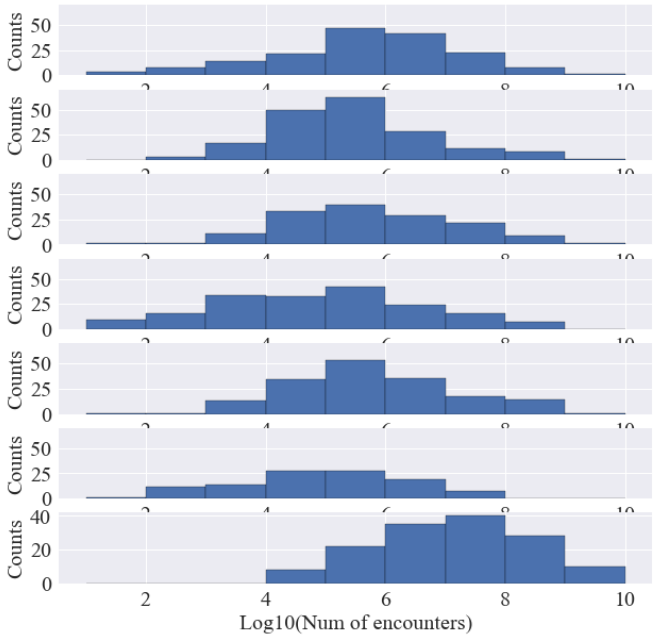

Figure 4.10 Distribution of planets according to the number of encounters experienced, for simulation set 1-7. The x axis represents the total number of close encounters each planets experienced throughout the simulation, and the y axis represents the total counts of planets in each bin of number of encounters.

planets migrated inward from 5 AU, the more eccentric they are, and the farther the planets migrated outward from 10 AU, the more eccentric they are too.

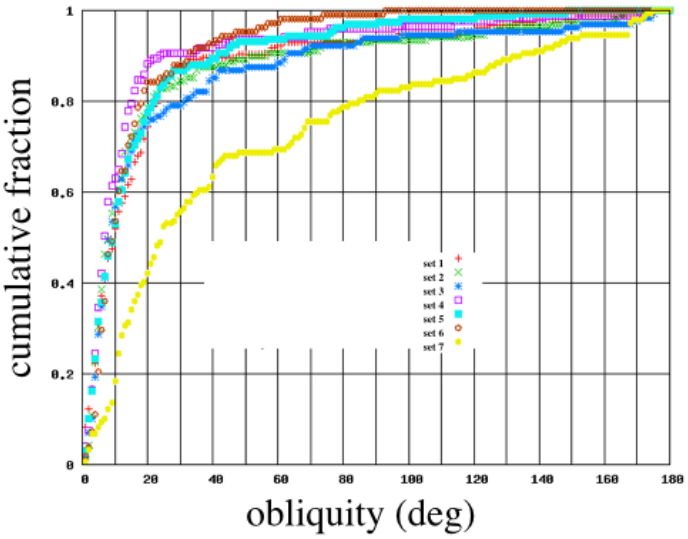

Figure 4.11 Cumulative distribution of final obliquity. The x axis represents obliquity and the y axis represents cumulative fraction of planets. Each colored line represents cumulative distribution of the planets in one simulation set. The obliquity distribution in set 7 clearly separates from all other simulation sets.

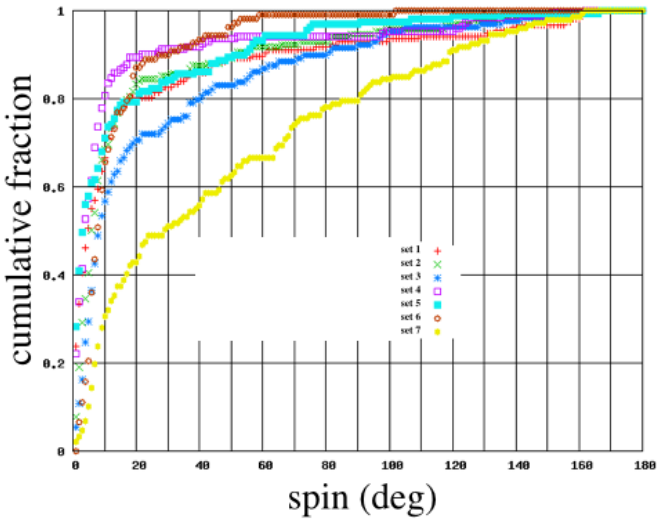

Figure 4.12 Cumulative distribution of final spin.

This demonstrates that planets with more radial migration are also more eccentric, which indicates that those planets experienced more planet-planet scattering. This outcome has close resemblance with the results obtained in past work

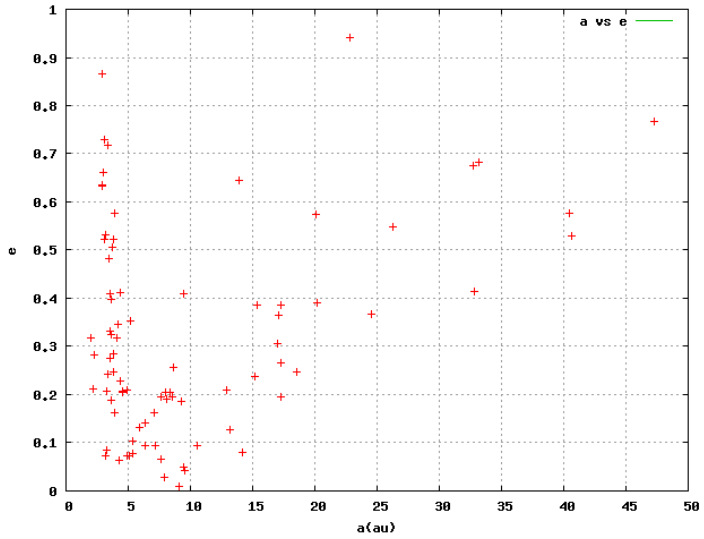

Figure 4.13 Scatter plot of planet final a and e for simulation set 1. Each red dot represent one simulated planet with a certain final a and e.

(Chatterjee et al., 2008, Jurić & Tremaine, 2008, Raymond et al., 2010). Figure 4.14 shows a similar trend for a $< 5AU$. The farther the planets migrated inward from 5 AU, the more inclined they become. Planets beyond 10 AU doesn't have a similar trend likely because star plays an important role in torquing up planet inclination, so the distant planets do not have significant inclination evolution contributed by the star. In turn, e do not show as clear a correlations with i, o, and s.

The final inclination of planet also has significant correlations with the final spin and obliquity. In figure 4.15, planet final inclination and obliquity show a positive correlation. The more inclined the planets have become, the more oblique they are. In reference to figure 4.16, the obliquity of planets are not all from the evolution of inclinations with a static spin, as spin and orbital inclination also have a significant positive correlation. The more inclined the planets are, the more evolved their spin and thus obliquities. These correlations are likely due to that the stellar torque plays a major role for the evolution of $i, s,$

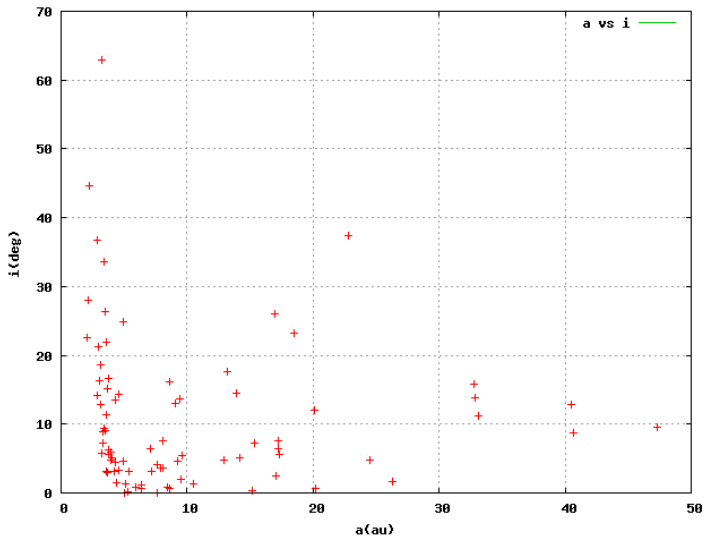

Figure 4.14 Scatter plot of planet final a and i for simulation set 1. Each red dot represent one simulated planet with a certain final a and i.

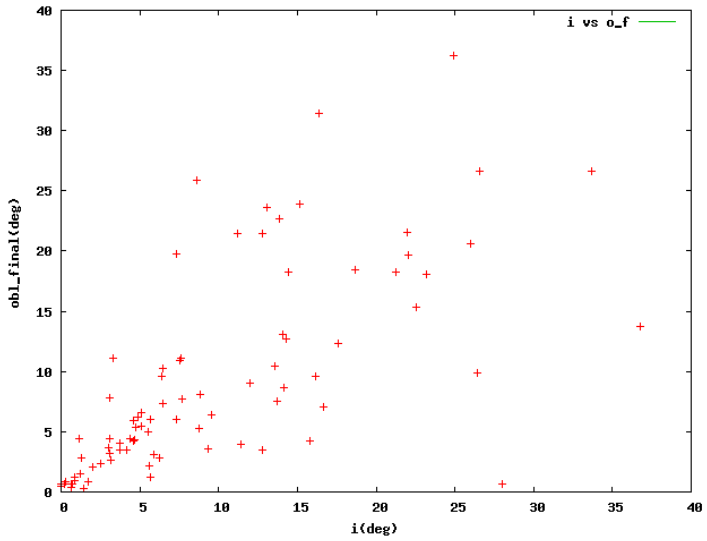

Figure 4.15 Scatter plot of planet final i and o for simulation set 1. Each red dot represent one simulated planet with a certain final i and o.

and o, which will be further discussed in the next paragraph.

Here we investigate the correlation between the minimum pericentric distance (q) a planet has experienced throughout the simulation, and the planet's s_{max}, o_{max}, and i_f. When such correlations are significant, it indicates that close-

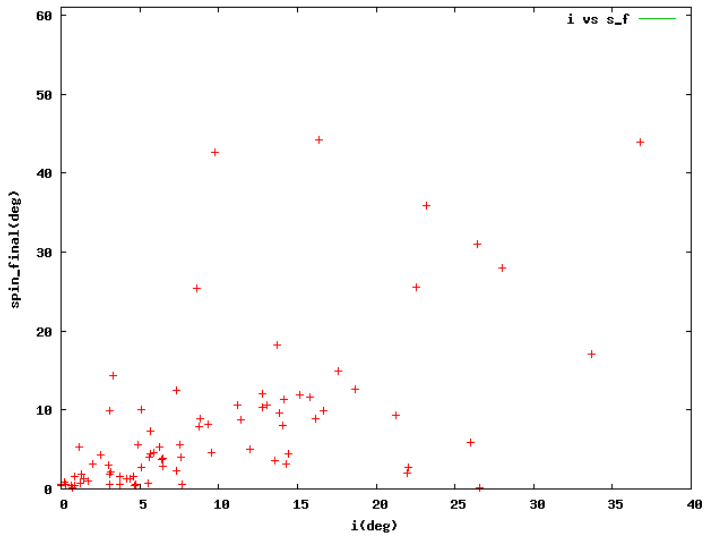

Figure 4.16 Scatter plot of planet final i and s for simulation set 1. Each red dot represent one simulated planet with a certain final i and s.

ness to the star contributes to the evolution of i, o and s, which in turn indicates that stellar torque is the most important driver for the evolution. Figure 4.17 shows a clear negative correlation between q_{min} and s_{max}. It also shows that to have significant spin evolution ($s_f > 40°$), it is required that the planet has once been as close to the star as at least 2 AU, in order to receive the torque required for significant spin evolution. Figure 4.18 and 4.19 shows correlation between q_{min} and o_{max}, and q_{min} and i_{max}. They have very similar stories to tell as figure 4.17. These again prove that stellar torque is the main contributor for the evolution of spin, orbital inclination, and thus obliquity.

Correlations between the planet's various orbital parameters in simulation 2-6 demonstrate similar trends as set 1. Planet initial orbit, stellar mass, planet torque contribution, and planet mass do not qualitatively change the nature and outcome of planet-planet scattering. However, such correlations in simulation set 7 are quite different. Since all planets have equal masses, simulation set 7 represents a singular case, therefore the difference. In figure 4.20 and 4.21,

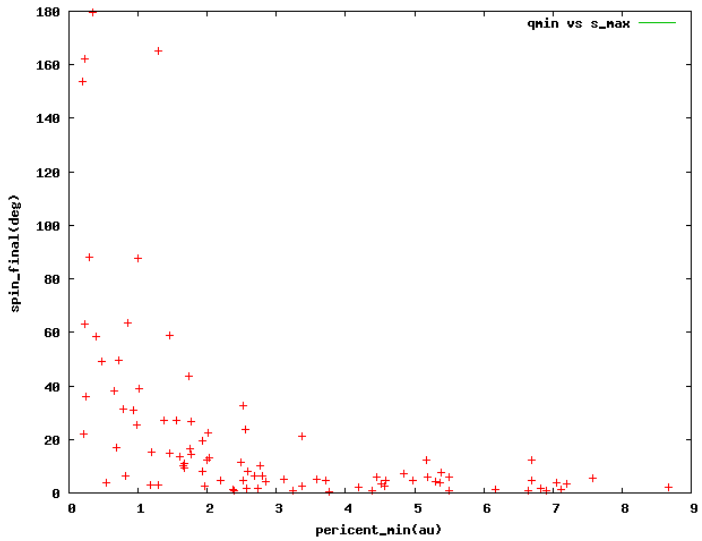

Figure 4.17 Scatter plot of planet minimum pericentric distance and maximum spin angle for simulation set 1. Each red dot represent one simulated planet with a certain q_{min} and s_{max}.

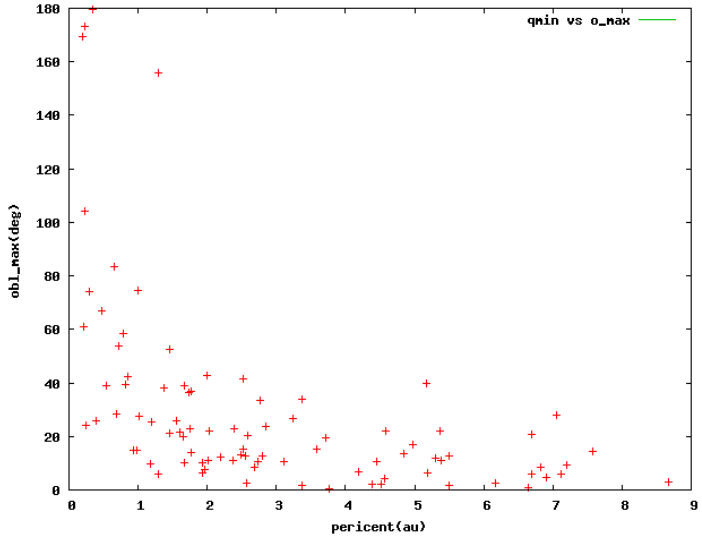

Figure 4.18 Scatter plot of planet minimum pericentric distance and maximum obliquity for simulation set 1. Each red dot represent one simulated planet with a certain q_{min} and o_{max}.

planets in simulation set 7 have much more evenly distributed scatters in the a, e, i orbital parameter spaces compared with planets in simulation set 1. Especially unique is the population with large semi-major axes and low eccentrici-

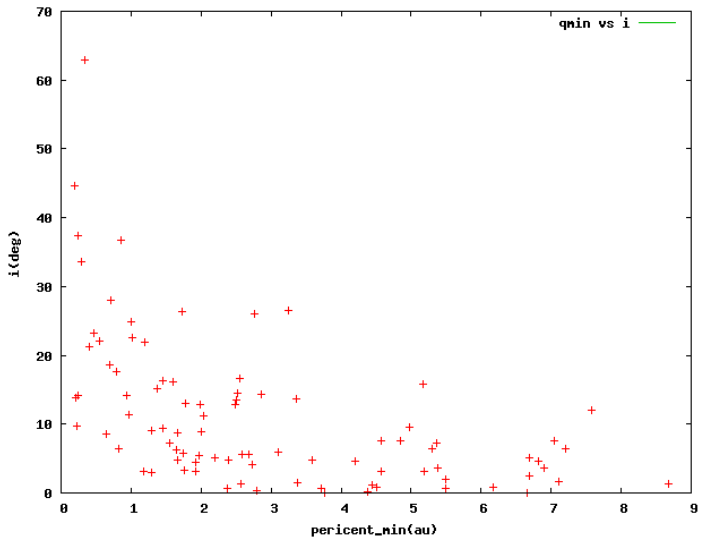

Figure 4.19 Scatter plot of planet minimum pericentric distance and maximum inclination for simulation set 1. Each red dot represent one simulated planet with a certain q_{min} and i_f.

ties. Planets can scatter far with their eccentricities staying mild, likely due to the difficulty to transfer significant amounts of orbital angular momentum between equal mass planets. A similar trend in the a-e and a-i scatter plot between set 1 and set 7 is a significant population of planets surviving close to the star (warm Jupiters) with a very wide distribution in eccentricities and inclinations. Nevertheless, in set 7, there exists retrograde planets, and in set 1 all close-in planets are on prograde orbits.

As with correlations between final $i, s,$ and o in simulation set 7, they are much weaker than set 1. In figure 4.22 and 4.23, the planets have more evenly distributed scatters in the $i, s,$ and o orbital parameter spaces, compared with planets in simulation set 1. Constant jitters for planetary orbits in equal mass planet scattering can contribute to the more chaotic and robust spin evolution of the planets.

The correlation between q_{min} and s_{max}, o_{max}, i_f in simulation set 7 are qualita-

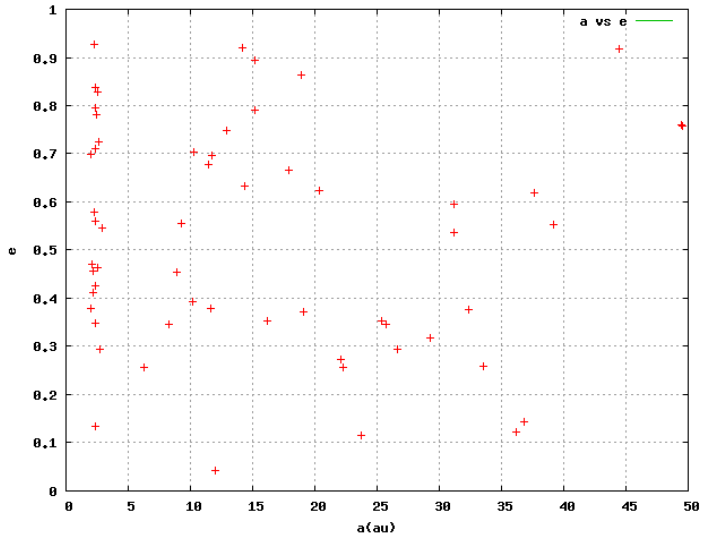

Figure 4.20 Scatter plot of planet final a and e for simulation set 7. Each red dot represent one simulated planet with a certain final a and e.

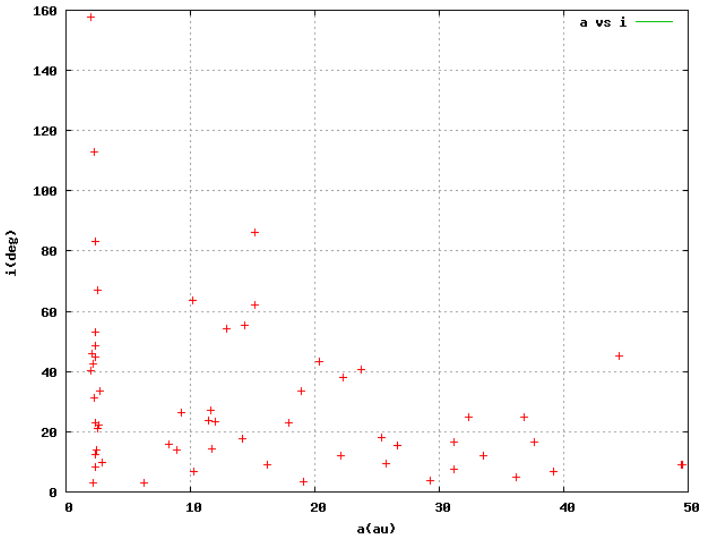

Figure 4.21 Scatter plot of planet final a and i for simulation set 7. Each red dot represent one simulated planet with a certain final a and i.

tively similar to simulation set 1. There're clear negative correlations. However, there's a significant difference in s_{max}, o_{max}, i_f for planets that have been super close to the star. They were tilted to retrograde inclination and obliquity ($i > 90°$ and $o > 90°$) by the star. In summary, correlations between q_{min} and s_{max}, o_{max}, i_f in all simulation sets are strong, ranging from -0.5 to -0.65 (fig. 4.27).

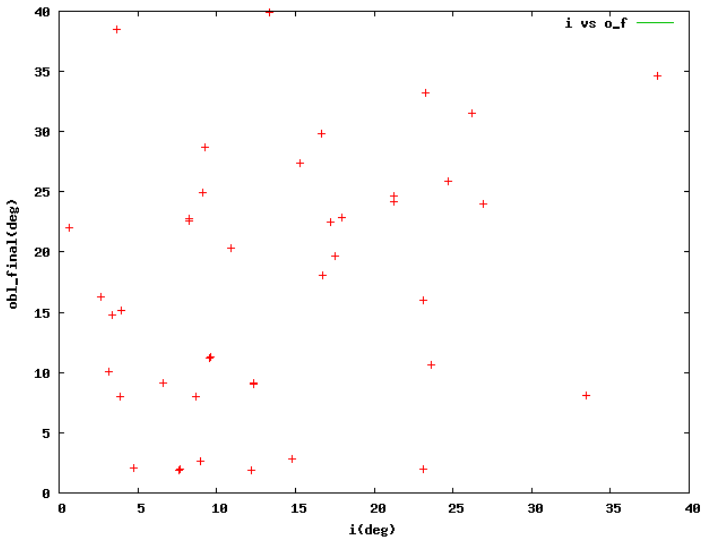

Figure 4.22 Scatter plot of planet final i and o for simulation set 7. Each red dot represent one simulated planet with a certain final i and o.

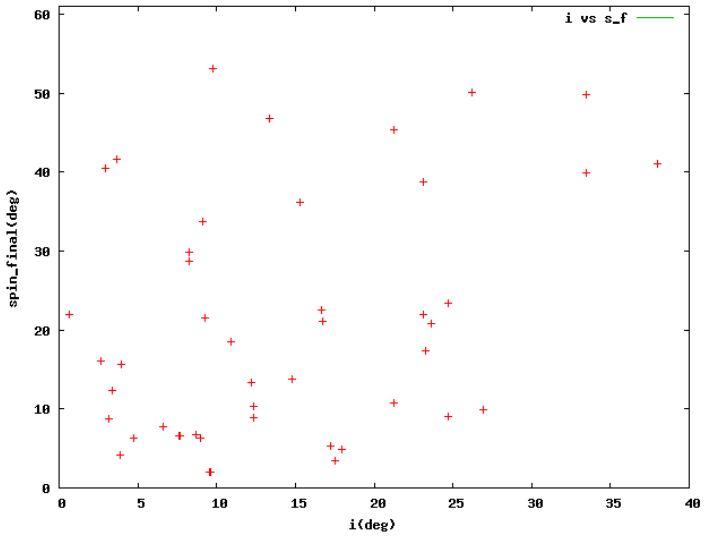

Figure 4.23 Scatter plot of planet final i and s for simulation set 7. Each red dot represent one simulated planet with a certain final i and s.

Correlations between final orbital parameters and planet masses are generally weak for all simulations. A much larger simulation set is likely necessary to further investigate such correlations. Figure 4.28 contains binned plots for a_f for planets in simulation set 1 with four different masses. Correlations of final semi-major axis with mass aren't very clear from the plots, partly due to a small

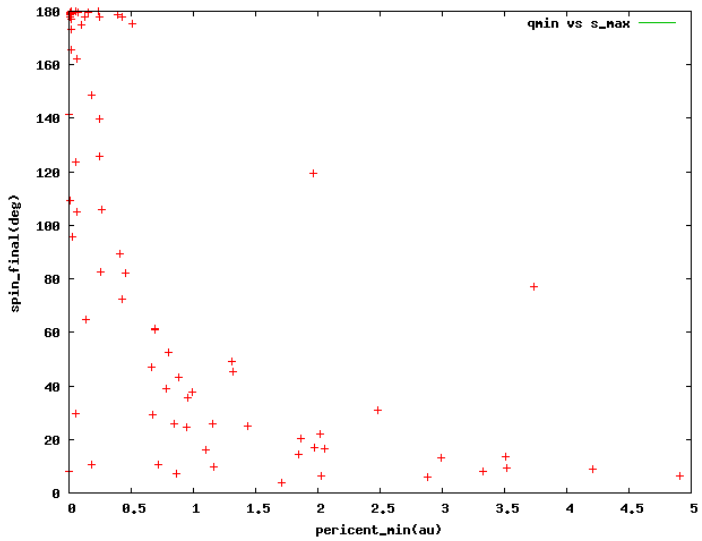

Figure 4.24 Scatter plot of planet minimum pericentric distance and maximum spin angle for simulation set 7. Each red dot represent one simulated planet with a certain q_{min} and s_{max}.

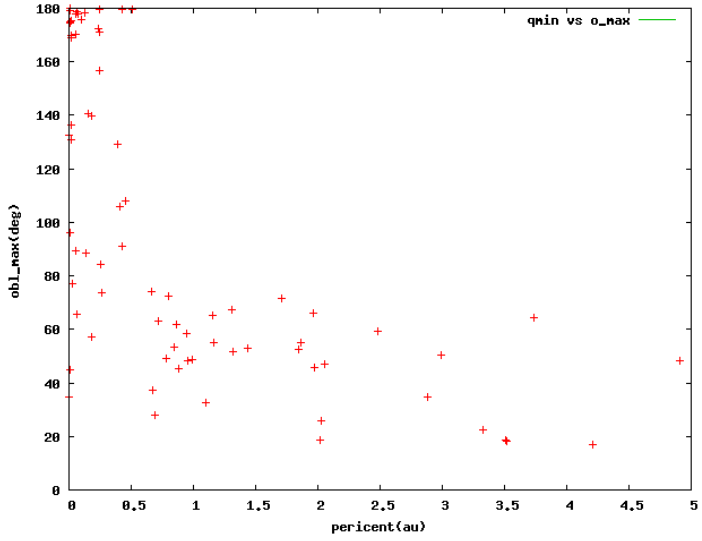

Figure 4.25 Scatter plot of planet minimum pericentric distance and maximum obliquity for simulation set 7. Each red dot represent one simulated planet with a certain q_{min} and o_{max}.

number of simulations conducted.

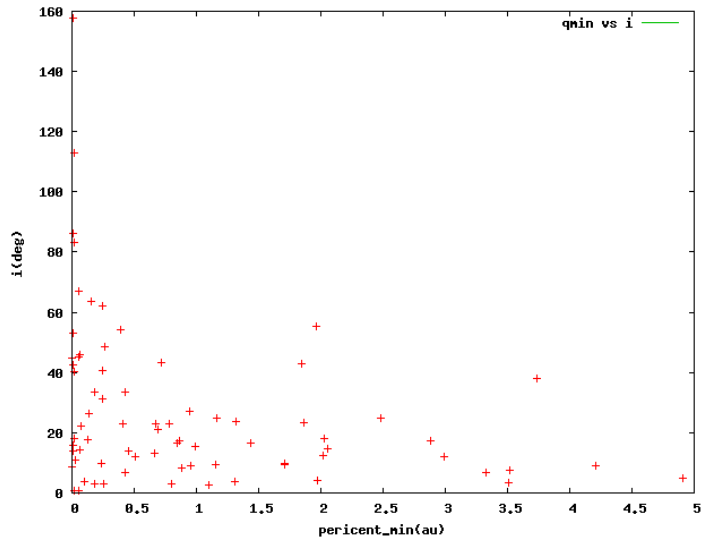

Figure 4.26 Scatter plot of planet minimum pericentric distance and maximum inclination for simulation set 7. Each red dot represent one simulated planet with a certain q_{min} and i_f.

sim set	q_{min} & o_{max}	q_{min} & s_{max}
1	-0.48	-0.48
2	-0.53	-0.50
3	-0.51	-0.57
4	-0.51	-0.47
5	-0.59	-0.55
6	-0.53	-0.53
7	-0.65	-0.65

Figure 4.27 Correlation parameters of q_{min} vs. o_{max} & q_{min} vs. s_{max} for simulation set 1-7.

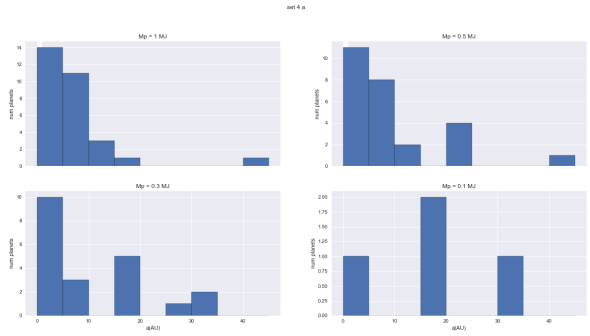

Figure 4.28 Distribution of planets according to their final semi-major axis for simulation set 1, for different planet masses.

4.5 Conclusion

Although planet-planet perturbation when the planets come close together is not very important for their spin evolution, due to the short duration of planetary close encounters, planet-planet scattering can still produce highly oblique planets. A close match between $\frac{d\hat{s}}{dt}$ and $\frac{d\hat{L}}{dt}$ can drive significant spin evolution and generate highly oblique planets. Planet-planet scattering can produce such required configurations since planets are constantly kicking each other and jumping around. This paper analyzed one particular simulation in depth and successfully verified the theoretical grounding for planet spin evolution.

Planet-planet scattering also provides another required condition to drive spin evolution of planets. Since the stellar torque is the main contributor to planet spin evolution, it is required that the planets have once come close to the star ($\sim< 2AU$). Planet-planet scattering is able to push planets closer in to the star by moving out or ejecting some of the planet members in the system, so it can drive planet obliquity evolution. The importance of stellar torque is demonstrated in multiple ways. Section 4.4 discusses torque contribution from other planets and the star on planet 0, in two close encounters and throughout

the entire simulation. It shows that although torque contribution from other planets are dramatically larger than from the star during planetary close encounters, the summed contribution of stellar torque throughout 1 million years is ~ 5000 times greater than of the other planets. Therefore, planet-planet torque can likely be neglected in planet-planet scattering for planet obliquity evolution. The strong correlation between q_{min} and o_{max}, s_{max} shows that closeness to the star drives spin evolution, reinforcing the hypothesis that stellar torque is the main contributor.

Planetary systems with all equal mass planets behave qualitatively differently than planetary systems with mixed mass distributions, since it represents a singular case. The much more robust and frequent interactions between the planets due to the difficulty in ejecting each other produces planets with much more excited orbits, with higher average semi-major axes, eccentricities, inclinations, and obliquities. The fraction of planets with retrograde orbital inclination and obliquity is nearly twice as much in equal mass systems compared with planetary systems with mixed mass distributions. The constant jitter of planetary orbits due to mutual perturbations in equal mass systems also complicates the behavior of spin, leading to more high obliquity planets.

In summary, planet-planet scattering is a viable mechanism for tilting planets, through resonance crossing and rough correspondence between $\frac{d\hat{s}}{dt}$ and $\frac{d\hat{L}}{dt}$. The scattered distribution of orbital elements with obliquity and their correlations, along with future observations of giant planet obliquity, can further verify planet-planet scattering as a planet formation model and distinguish the origin of some populations of planets.

CHAPTER 5

CONCLUSION

In addition to the observed diverse physical properties among extrasolar planets, the dynamics of extrasolar moons and giant planets are similarly diverse. Extrasolar planetary systems have drastically different orbits compared to our Solar System, which is an indicating that the formation and evolution paths can be more complex and diverse. Planet-planet scattering provides strong and rapidly changing perturbations to moons and planets, leading to versatile dynamical behaviors not observed within the Solar System. In the first part of this work, we explored the dynamical instability that can happen to close-in satellites when planet oblateness is not accounted for in non-coplanar multiplanet systems. Simulations include two secularly interacting Jupiter-mass planets mutually inclined by 10°, with the host planet either oblate or spherical. With a spherical host planet, moons within a critical planetocentric distance experience high inclinations and in some cases high eccentricities, while more distant moons orbit stably with low inclinations and eccentricities, as expected. These counter-intuitive dynamical phenomena disappear with an oblate host planet, in which case the moons' Laplace plane transitions from the host planet's equatorial plane to the host planet's precessing orbital plane as their semi-major axes increase, and all moons are dynamically stable with very mild changes in orbits. Direct perturbation from the perturbing planet has been investigated and ruled out as an explanation for the behavior of the innermost satellites, therefore leaving the central star's perturbation as the cause. Instability occurs while the nodal precession of the satellite and the central star (as seen from the host planet's frame) approaches the 1:1 secular resonance. In non-coplanar systems, around a non-oblate planet, the nodal precession of the

moon becomes slow and comparable to that of the planet, giving rise to resonant configurations. The above effect needs to be taken into account in setting up numerical simulations.

In the second and third part of this work, we investigate unexplored celestial populations and orbital parameters - extrasolar moons and giant planet obliquity - in the context of planet-planet scattering. Planet-planet scattering is the leading mechanism to explain the broad eccentricity distribution of observed giant exoplanets. In the second part of this work, we study the orbital stability of primordial giant planet moons in this scenario. We use N-body simulations including realistic oblateness and evolving spin evolution for the giant planets. We find that the vast majority ($\sim$ 80-90% across all our simulations) of orbital parameter space for moons is destabilized. There is a strong radial dependence, as moons past $\sim 0.1 R_{Hill}$ are systematically removed. Closer-in moons on Galilean-moon-like orbits ($<$ 0.04 R_{Hill}) have a good ($\sim$ 20-40%) chance of survival. Destabilized moons may undergo a collision with the star or a planet, be ejected from the system, be captured by another planet, be ejected but still orbiting its free-floating host planet, or survive on heliocentric orbits as "planets". The survival rate of moons increases with the host planet mass but is independent of the planet's final (post-scattering) orbits. Based on our simulations we predict the existence of an abundant galactic population of free-floating (former) moons.

In the third part of this work, we further apply the planet-planet scattering model in studying planet obliquity evolution - a new orbital parameter never treated in theoretical literature that is ready to be observed. Tilting planets are possible with secular perturbations. Resonances between the spin precession

and secular frequencies in the system can make the spin orientation evolve. Here we deviate from secular problems and study the possibility of tilting giant planets during planetary close encounters. It turns out to be a viable mechanism for tilting planets and even produce retrograde obliquities for planets. In systems with mixed planet masses, $\sim 10-15\%$ of simulated planets survived on high obliquities $> 40^\circ$. In systems with equal planet mass, a very high fraction of $\sim 35-40\%$ of simulated planets survived on obliquities $> 40\,\mathrm{deg}$. On average, due to more planet-planet interactions in equal mass planetary systems, the planets end up with more excited orbits than systems with mixed planet mass. Stellar torque plays the most important role in the obliquity / spin evolution of the planets. There exists a clear negative correlation between the planets' minimum pericenter passage to the star and their obliquities. Planets with higher obliquities have came closer to the star. Planet-planet torque is significantly higher than stellar torque during close encounters between the planets, but due to the short duration of close encounters, the summed contribution of planet torque is 3 to 4 orders smaller than stellar torque, so planet torque is negligible. Besides the stellar torque, a correspondence between $\frac{d\hat{s}}{dt}$ and $\frac{d\hat{L}}{dt}$ is required for significant spin evolution. When the two quantities are orders of magnitudes different, the planets retain a constant obliquity or a staticaly precessing spin. When they are roughly equal or through resonance crossing between the two quantities, spin can evolve chaotically. The distribution of planets in the parameter space of various orbital elements such as o, s, i, a, e can be a good constraint for whether future observed population of planets form from planet-planet scattering.

In a word, theoretical exploration of extrasolar systems is strikingly interesting for their immense possibilities and diversity. Future observations of ex-

trasolar moons with microlensing by WFIRST, and observations with transit, astrometry, and radial velocity methods can all possibly bring us new insights on planet and moon formation, and of course a world of wonders. With its capability for detecting free-floaters down to Ganymede mass, WFIRST is likely to reveal the exciting existence of free-floating exomoons. If the mass function of observed free-floaters by WIFRST differs from the current observed planets, with an excess at the lower mass tail, it can imply the existence of free-floating moons. With observations of bound moons and free-floating moons, WFIRST can also testify the theoretical prediction in this work on the fraction of free-floating moons and bound moons. Observations of planet obliquity can serve as a good test against planet formation model.

BIBLIOGRAPHY

Adams, F. C., & Laughlin, G. 2003, , 163, 290

Agol, E., Steffen, J., Sari, R., & Clarkson, W. 2005, MNRAS, 359, 567

Anderson, K. R., Storch, N. I., & Lai, D. 2016, MNRAS, 456, 3671

Barnes, J. W., & O'Brien, D. P. 2002, ApJ, 575, 1087

Barnes, R., & Quinn, T. 2004, ApJ, 611, 494

Bennett, D. P., Batista, V., Bond, I. A., et al. 2014, ApJ, 785, 155

Bennett, D. P., & Rhie, S. H. 1996, ApJ, 472, 660

—. 2002, ApJ, 574, 985

Bennett, D. P., Akeson, R., Anderson, J., et al. 2018, arXiv e-prints, arXiv:1803.08564

Bills, B. G. 1990, J. Geophys. Res., 95, 14137

—. 2005, , 175, 233

Brasser, R., Morbidelli, A., Gomes, R., Tsiganis, K., & Levison, H. F. 2009, A&A, 507, 1053

Canup, R. M., & Ward, W. R. 2009, Origin of Europa and the Galilean Satellites, ed. R. T. Pappalardo, W. B. McKinnon, & K. K. Khurana, 59

Carrera, D., Davies, M. B., & Johansen, A. 2016, MNRAS, 463, 3226

Chambers, J. E. 1999, MNRAS, 304, 793

Chambers, J. E., Wetherill, G. W., & Boss, A. P. 1996, , 119, 261

Chatterjee, S., Ford, E. B., Matsumura, S., & Rasio, F. A. 2008, ApJ, 686, 580

Chirikov, B. V. 1979, Phys. Rep., 52, 263

Cumming, A., Butler, R. P., Marcy, G. W., et al. 2008, PASP, 120, 531

Deienno, R., Yokoyama, T., Nogueira, E. C., Callegari, N., & Santos, M. T. 2011, A&A, 536, A57

Dobos, V., Heller, R., & Turner, E. L. 2017, A&A, 601, A91

Domingos, R. C., Winter, O. C., & Yokoyama, T. 2006, MNRAS, 373, 1227

Donnison, J. R. 2010, MNRAS, 406, 1918

Ford, E. B., & Rasio, F. A. 2008, ApJ, 686, 621

Forgan, D., & Kipping, D. 2013, MNRAS, 432, 2994

Frouard, J., & Yokoyama, T. 2013, Celestial Mechanics and Dynamical Astronomy, 115, 59

Goldreich, P. 1965, AJ, 70, 5

Gomes, R., Levison, H. F., Tsiganis, K., & Morbidelli, A. 2005, Nature, 435, 466

Gong, Y.-X., Zhou, J.-L., Xie, J.-W., & Wu, X.-M. 2013, ApJ, 769, L14

Guillochon, J., Ramirez-Ruiz, E., & Lin, D. 2011, ApJ, 732, 74

Han, C. 2008, ApJ, 684, 684

Han, C., & Han, W. 2002, ApJ, 580, 490

Heller, R. 2012, A&A, 545, L8

—. 2014, ApJ, 787, 14

Heller, R., & Barnes, R. 2013, Astrobiology, 13, 18

—. 2015, International Journal of Astrobiology, 14, 335

Heller, R., & Zuluaga, J. I. 2013, ApJ, 776, L33

Heller, R., williams, D., Kipping, D., et al. 2014, Astrobiology, 14, 798

Hilton, J. L. 1991, AJ, 102, 1510

Hinkel, N. R., & Kane, S. R. 2013, ApJ, 774, 27

Holman, M. J., & Murray, N. W. 2005, Science, 307, 1288

Holman, M. J., & Wiegert, P. A. 1999, AJ, 117, 621

Hong, Y.-C., Raymond, S. N., & Lunine, J. I. 2012, AAS/DPS, 44, 100.06

Hong, Y.-C., Tiscareno, M. S., Nicholson, P. D., & Lunine, J. I. 2015, Monthly Notices of the Royal Astronomical Society, 449, 828. https://doi.org/10.1093/mnras/stv311

Jeans, J. H. 1919, MNRAS, 79, 408

Jewitt, D., & Haghighipour, N. 2007, ARA&A, 45, 261

Johnson, J. A., Butler, R. P., Marcy, G. W., et al. 2007, ApJ, 670, 833

Jurić, M., & Tremaine, S. 2008, ApJ, 686, 603

Kaltenegger, L. 2010, ApJ, 712, L125

Kane, S. R. 2017, ApJ, 839, L19

Kasting, J. F., Whitmire, D. P., & Reynolds, R. T. 1993, , 101, 108

Kinoshita, H., & Nakai, H. 1991, Celestial Mechanics and Dynamical Astronomy, 52, 293

Kipping, D. M. 2009, MNRAS, 392, 181

Kipping, D. M., Bakos, G. Á., Buchhave, L., Nesvorný, D., & Schmitt, A. 2012, ApJ, 750, 115

Kipping, D. M., Fossey, S. J., & Campanella, G. 2009, MNRAS, 400, 398

Kipping, D. M., Schmitt, A. R., Huang, X., et al. 2015, ApJ, 813, 14

Kouwenhoven, M. B. N., Goodwin, S. P., Parker, R. J., et al. 2010, MNRAS, 404, 1835

Levison, H. F., Morbidelli, A., Van Laerhoven, C., Gomes, R., & Tsiganis, K. 2008, , 196, 258

Lin, D. N. C., & Ida, S. 1997, ApJ, 477, 781

Lovis, C., & Mayor, M. 2007, A&A, 472, 657

Lucas, P. W., & Roche, P. F. 2000, Monthly Notices of the Royal Astronomical Society, 314, 858. https://doi.org/10.1046/j.1365-8711.2000.03515.x

Marzari, F., & Weidenschilling, S. J. 2002, , 156, 570

Matsumura, S., Ida, S., & Nagasawa, M. 2013, ApJ, 767, 129

Mayor, M., Marmier, M., Lovis, C., et al. 2011, arXiv e-prints, arXiv:1109.2497

Morbidelli, A., Brasser, R., Gomes, R., Levison, H. F., & Tsiganis, K. 2010, AJ, 140, 1391

Morbidelli, A., Levison, H. F., Tsiganis, K., & Gomes, R. 2005, Nature, 435, 462

Mróz, P., Udalski, A., Skowron, J., et al. 2017, Nature, 548, 183

Murray, C. D., & Dermott, S. F. 1999, Solar system dynamics

Namouni, F. 2010, The Astrophysical Journal, 719, L145. https://doi.org/10.1088%2F2041-8205%2F719%2F2%2Fl145

Nesvorný, D., & Vokrouhlický, D. 2009, AJ, 137, 5003

Nesvorný, D., Vokrouhlický, D., & Morbidelli, A. 2007, AJ, 133, 1962

Nicholson, P. D., Cuk, M., Sheppard, S. S., Nesvorny, D., & Johnson, T. V. 2008, Irregular Satellites of the Giant Planets, ed. M. A. Barucci, H. Boehnhardt, D. P. Cruikshank, A. Morbidelli, & R. Dotson, 411–424

Payne, M. J., Deck, K. M., Holman, M. J., & Perets, H. B. 2013, ApJ, 775, L44

Peters, M. A., & Turner, E. L. 2013, ApJ, 769, 98

Pu, B., & Lai, D. 2018, Monthly Notices of the Royal Astronomical Society, 478, 197. https://doi.org/10.1093/mnras/sty1098

Quinn, T. R., Tremaine, S., & Duncan, M. 1991, AJ, 101, 2287

Rasio, F. A., & Ford, E. B. 1996, Science, 274, 954

Raymond, S. N., Armitage, P. J., & Gorelick, N. 2009, ApJ, 699, L88

—. 2010, ApJ, 711, 772

Raymond, S. N., & Izidoro, A. 2017, , 297, 134

Raymond, S. N., Armitage, P. J., Moro-Martín, A., et al. 2011, A&A, 530, A62

—. 2012, A&A, 541, A11

Reynolds, R. T., McKay, C. P., & Kasting, J. F. 1987, Advances in Space Research, 7, 125

Rowan, D., Meschiari, S., Laughlin, G., et al. 2016, ApJ, 817, 104

Sartoretti, P., & Schneider, J. 1999, A&AS, 134, 553

Sasaki, T., Barnes, J. W., & O'Brien, D. P. 2012, ApJ, 754, 51

Scharf, C. A. 2006, ApJ, 648, 1196

Simon, A., Szatmáry, K., & Szabó, G. M. 2007, A&A, 470, 727

Spalding, C., Batygin, K., & Adams, F. C. 2016, ApJ, 817, 18

Spergel, D., Gehrels, N., Baltay, C., et al. 2015, arXiv e-prints, arXiv:1503.03757

Storch, N. I., Anderson, K. R., & Lai, D. 2014, Science, 345, 1317

Storch, N. I., & Lai, D. 2015, MNRAS, 448, 1821

Sumi, T., Kamiya, K., Bennett, D. P., et al. 2011, Nature, 473, 349

Szulágyi, J. 2017, The Astrophysical Journal, 842, 103. https://doi.org/10.3847%2F1538-4357%2Faa7515

Szulágyi, J., Mayer, L., & Quinn, T. 2017, MNRAS, 464, 3158

Szulágyi, J., Plas, G. v. d., Meyer, M. R., et al. 2018, MNRAS, 473, 3573

Teachey, A., & Kipping, D. M. 2018, Science Advances, 4, https://advances.sciencemag.org/content/4/10/eaav1784.full.pdf. https://advances.sciencemag.org/content/4/10/eaav1784

Tinney, C. G., Wittenmyer, R. A., Butler, R. P., et al. 2011, ApJ, 732, 31

Tremaine, S. 1993, in Astronomical Society of the Pacific Conference Series, Vol. 36, Planets Around Pulsars, ed. J. A. Phillips, S. E. Thorsett, & S. R. Kulkarni, 335–344

Tremaine, S., Touma, J., & Namouni, F. 2009, AJ, 137, 3706

Tsiganis, K., Gomes, R., Morbidelli, A., & Levison, H. F. 2005, Nature, 435, 459

Vanderburg, A., Rappaport, S. A., & Mayo, A. W. 2018, AJ, 156, 184

Veras, D., & Armitage, P. J. 2004, , 172, 349

—. 2005, ApJ, 620, L111

Veras, D., & Raymond, S. N. 2012, MNRAS, 421, L117

Ward, W. R. 1973, Science, 181, 260

—. 1974, J. Geophys. Res., 79, 3375

Weidenschilling, S. J., & Marzari, F. 1996, Nature, 384, 619

Williams, D. M., Kasting, J. F., & Wade, R. A. 1997, Nature, 385, 234

Winn, J. N., & Fabrycky, D. C. 2015, Annual Review of Astronomy and Astrophysics, 53, 409. https://doi.org/10.1146/annurev-astro-082214-122246

Wittenmyer, R. A., Butler, R. P., Tinney, C. G., et al. 2016, ApJ, 819, 28

Zapatero Osorio, M. R., Béjar, V. J. S., Martın, E. L., et al. 2000, Science, 290, 103. https://science.sciencemag.org/content/290/5489/103

CHAPTER 6

APPENDIX: N-BODY INTEGRATOR - BULIRSCH STOER ALGORITHM

```
c
   %%%%
```

```
     drift' or 'orbel'.
c
c The standard symplectic (MVS) algorithm is described in
     J.Widsom and
c M.Holman (1991) Astronomical Journal, vol 102, pp1528.
c
c The hybrid symplectic algorithm is described in J.E.
     Chambers / Yu-Cian Hong (1999)
c Monthly Notices of the RAS, vol 304, pp793.
c
c RADAU is described in E.Everhart (1985) in ''The
     Dynamics of Comets:
c Their Origin and Evolution'' p185-202, eds. A.Carusi & G
     .B.Valsecchi,
c pub. Reidel.
c
c The Bulirsch-Stoer algorithms are described in W.H.Press
      et al. (1992)
c ''Numerical Recipes in Fortran'', pub. Cambridge.
c
     ------------------------------------------------------------------------
c
c Variables:
c ----------
c  M        = mass (in solar masses)
```

```
c  XH     = coordinates (x,y,z) with respect to the
   central body (AU)
c  VH     = velocities (vx,vy,vz) with respect to the
   central body (AU/day)
c  S      = spin angular momentum (solar masses AU^2/day)
c  RHO    = physical density (g/cm^3)
c  RCEH   = close-encounter limit (Hill radii)
c  STAT   = status (0 => alive, <>0 => to be removed)
c  ID     = name of the object (8 characters)
c  CE     = close encounter status
c  NGF    = (1-3) cometary non-gravitational (jet) force
   parameters
c   "     =  (4)  beta parameter for radiation pressure
   and P-R drag
c  EPOCH  = epoch of orbit (days)
c  NBOD   = current number of bodies (INCLUDING the central
    object)
c  NBIG   =    "       "    " big bodies (ones that perturb
    everything else)
c  TIME   = current epoch (days)
c  TOUT   = time of next output evaluation
c  TDUMP  = time of next data dump
c  TFUN   = time of next periodic effect (e.g. next check
   for ejections)
c  H      = current integration timestep (days)
c  EN(1)  = initial energy of the system
```

```
c  " (2) = current    "    " "    "
c  " (3) = energy change due to collisions, ejections etc.
c  AM(1,2,3) = as above but for angular momentum
c
c Integration Parameters :
c ------------------------
c  ALGOR = 1  -> Mixed-variable symplectic
c          2  -> Bulirsch-Stoer integrator
c          3  ->        "           "      (conservative
   systems only)
c          4  -> RA15 'radau' integrator
c          10 -> Hybrid MVS/BS (democratic-heliocentric
   coords)
c          11 -> Close-binary hybrid (close-binary coords
   )
c          12 -> Wide-binary hybrid (wide-binary coords)
c
c TSTART = epoch of first required output (days)
c TSTOP  =   "      final required output ( "  )
c DTOUT  = data output interval           ( "  )
c DTDUMP = data-dump interval             ( "  )
c DTFUN  = interval for other periodic effects (e.g. check
    for ejections)
c  H0    = initial integration timestep (days)
c  TOL   = Integrator tolerance parameter (approx. error
   per timestep)
```

```
c  RMAX  = heliocentric distance at which objects are
   considered ejected (AU)
c  RCEN  = radius of central body (AU)
c  JCEN(1,2,3) = J2,J4,J6 for central body (units of RCEN^
   i for Ji)
c
c Options:
c  OPT(1) = close-encounter option (0=stop after an
   encounter, 1=continue)
c  OPT(2) = collision option (0=no collisions, 1=merge, 2=
   merge+fragment)
c  OPT(3) = time style (0=days 1=Greg.date 2/3=days/years
   w/respect to start)
c  OPT(4) = o/p precision (1,2,3 = 4,9,15 significant
   figures)
c  OPT(5) = < Not used at present >
c  OPT(6) = < Not used at present >
c  OPT(7) = apply post-Newtonian correction? (0=no, 1=yes)
c  OPT(8) = apply user-defined force routine mfo_user? (0=
   no, 1=yes)
c
c File variables :
c ---------------
c  OUTFILE  (1) = osculating coordinates/velocities and
   masses
c     "     (2) = close encounter details
```

```
c     "      (3) = information file
c  DUMPFILE (1) = Big-body data
c     "      (2) = Small-body data
c     "      (3) = integration parameters
c     "      (4) = restart file
c
c Flags :
c -----
c  NGFLAG = do any bodies experience non-grav. forces?
c                              ( 0 = no non-grav forces)
c                                1 = cometary jets only
c                                2 = radiation pressure/P-R
   drag only
c                                3 = both
c  OPFLAG = integration mode (-2 = synchronising epochs)
c                               -1 = integrating towards
   start epoch
c                                0 = main integration,
   normal output
c                                1 = main integration, full
   output
c
c
   ------------------------------------------------------------------------


c
```

```
      implicit none
      include 'mercury.inc'
c
      integer j,algor,nbod,nbig,opt(8),stat(NMAX),lmem(
     &   NMESS)
      integer opflag,ngflag,ndump,nfun
      real*8 m(NMAX),xh(3,NMAX),vh(3,NMAX),s(3,NMAX),rho(
     &   NMAX)
      real*8 dsdt(3,NMAX)
      real*8 rceh(NMAX),epoch(NMAX),ngf(4,NMAX),rmax,rcen,
     &   jcen(3)
      real*8 cefac,time,tstart,tstop,dtout,h0,tol,en(3),am
     &   (3)
      character*8 id(NMAX)
      character*80 outfile(3), dumpfile(4), mem(NMESS)
      external mdt_bs1, mdt_bs2
      external mco_h2b,mco_iden
c
      data opt/0,1,1,2,0,1,0,0/
c
c
      ------------------------------------------------------------------------
c
c Get initial conditions and integration parameters
      call mio_in (time,tstart,tstop,dtout,algor,h0,tol,
```

```
     rmax,rcen,jcen,
     %  en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,
        rceh,stat,id,
     %  epoch,ngf,opt,opflag,ngflag,outfile,dumpfile,lmem,
        mem)
c
c If this is a new integration, integrate all the objects
   to a common epoch.
      if (opflag.eq.-2) then
  20     open (23,file=outfile(3),status='old',access='
     append',err=20)
         write (23,'(/,a)') mem(55)(1:lmem(55))
         write (*,'(a)') mem(55)(1:lmem(55))
         call mxx_sync (time,tstart,h0,tol,jcen,nbod,nbig,m
            ,xh,vh,s,rho,
     %     rceh,stat,id,epoch,ngf,opt,ngflag)
         write (23,'(/,a,/)') mem(56)(1:lmem(56))
         write (*,'(a)') mem(56)(1:lmem(56))
         opflag = -1
         close (23)
      end if
c
c Main integration
      if (algor.eq.2) call mal_hvar (time,tstart,tstop,
         dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,
```

```
     nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,
     dumpfile,mem,
     %  lmem,mdt_bs1,dsdt)
c
      if (algor.eq.3) call mal_hvar (time,tstart,tstop,
     dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,
     nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,
     dumpfile,mem,
     %  lmem,mdt_bs2,dsdt)
c
c Do a final data dump
      do j = 2, nbod
        epoch(j) = time
      end do
      call mio_dump (time,tstart,tstop,dtout,algor,h0,tol,
     jcen,rcen,
     %  rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     rho,rceh,stat,
     %  id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
c
c Calculate and record the overall change in energy and
   ang. momentum
  50  open  (23, file=outfile(3), status='old', access='
```

```
      append',
     %  err=50)
       write (23,'(/,a)') mem(57)(1:lmem(57))
       call mxx_en (jcen,nbod,nbig,m,xh,vh,s,en(2),am(2))
c
       write (23,231) mem(58)(1:lmem(58)),
     %  abs((en(2) + en(3) - en(1)) / en(1))
       write (23,232) mem(59)(1:lmem(59)),
     %  abs((am(2) + am(3) - am(1)) / am(1))
       write (23,231) mem(60)(1:lmem(60)), abs(en(3) / en
          (1))
       write (23,232) mem(61)(1:lmem(61)), abs(am(3) / am
          (1))
       close (23)
       write (*,'(a)') mem(57)(1:lmem(57))
c
c
     ------------------------------------------------------------------
c
 231   format (/,a,1p1e12.5)
 232   format (a,1p1e12.5)
       stop
       end
cc
     %%%%
```

```
c
c       SPIN.FOR (May 2019)
c
c Author: Yu-Cian Hong
c
%%%
```

```
      real*8 v2,r,temp,e,p,q,aj,dpdt,dqdt,h_leni,st(3,NMAX
     ),pt(3,NMAX)
      real*8 rr,cc,bb,aa,peri,maj,tt,alpha,xyz_tmp(3),
     uvw_tmp(3)
      real*8 dPadt,dpsidt,tmp4,tmp5,ss(3),rsp(3,NMAX),rpp
     (3,NMAX,NMAX)
      real*8 hh(3),yy(3),sss(3),ll(3),tmpx,tmpy,tmpz
      real*8 dsxdt,dsydt,dszdt,equin_tmp,obl_tmp,xyz_rot
     (3)
      real*8 ls,ll_len,ss_len,incl_tmp,node_tmp,sini,sini2
     ,l1,l2,r3tmp
      real*8 L_orbit(3),arr_tmp1(3),arr_tmp2(3),arr_1(3),
     arr_2(3)
      real*8, DIMENSION(NMAX) :: psi = 0.,Pa=0.,sx = 0.,sy
     = 0.,sz = 1.
c
     ------------------------------------------------------------------

      r_j = 0.000462392945
c     scaled mass of jupiter, m(1) is scaled in cgs units
      m_j = 9.5478951698e-04*m(1)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     calculate solar torque & spin evolution (+e+a)
c
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      do j=2,nbig

c       initialize / reset ds/dt to zero for each pl at
   each time step
        dsdt(1,j) = 0.d0
        dsdt(2,j) = 0.d0
        dsdt(3,j) = 0.d0
c
        hx  = x(2,j) * v(3,j) - x(3,j) * v(2,j)
        hy  = x(3,j) * v(1,j) - x(1,j) * v(3,j)
        hz  = x(1,j) * v(2,j) - x(2,j) * v(1,j)
c
        h2  = hx**2.d0 + hy**2.d0 + hz**2.d0
        h1  = dsqrt (h2)
        hh  = [hx,hy,hz] / h1
c
        r2 = x(1,j)**2. + x(2,j)**2. + x(3,j)**2.
        r1 = dsqrt(r2)
c
c       normalize r_star-pl
        rsp(1,j) = -x(1,j)/r1
        rsp(2,j) = -x(2,j)/r1
        rsp(3,j) = -x(3,j)/r1
c       normalize S
        sx(j) = s(1,j)*1.d0
```

```
      sy(j) = s(2,j)*1.d0
      sz(j) = s(3,j)*1.d0
c       if (s(1,j).ne.0..and.s(2,j).ne.0..and.s(3,j).ne
     .0.) then
      tmp1 = (sx(j)**2. + sy(j)**2. +sz(j)**2.)*1.d0
      tmp2 = dsqrt(tmp1)*1.d0
      sx(j) = sx(j) /tmp2*1.d0
      sy(j) = sy(j) /tmp2*1.d0
      sz(j) = sz(j) /tmp2*1.d0
c       alpha_s = 3*K2(=G in Gaussian units)*J2*(C/MR2)/
     spin rate =
c       3*2.959122082855911d-4*0.0147*0.25/2.418 = 1.34922
     d-6
c       assuming Jupiter like parameters
c       Use https://arxiv.org/pdf/1109.1627.pdf for
     Jupiter moment of
c       inertia lambda (C / MR2) ~ 0.25
c        needs to be twice smaller
c         alpha_s = 3.95d-7/r1**3.  still too large
c        alpha_s = 3.81d-7/r1**3.
      alpha_s = 3.4626d-6/r1**3.*1.d0
c
c       s^ dot r^
      tmp1 = (rsp(1,j) * sx(j) + rsp(2,j) * sy(j)+rsp(3,
        j)*sz(j))*1.d0
      dsdt(1,j) = alpha_s * tmp1 *(sy(j)*rsp(3,j)-sz(j)*
```

```
     rsp(2,j))*1.d0
     dsdt(2,j) = alpha_s * tmp1 *(sz(j)*rsp(1,j)-sx(j)*
     rsp(3,j))*1.d0
     dsdt(3,j) = alpha_s * tmp1 *(sx(j)*rsp(2,j)-sy(j)*
     rsp(1,j))*1.d0
c
c
c    add torque from other planets
     do k = 2,nbig
      if (k.gt.j) then
c       perturbee distance from star
c       perturber distance from perturbee
       tmp2 = ((x(1,k)-x(1,j))**2. + (x(2,k)-x(2,j))
     **2.)*1.d0
       tmp2 = (tmp2 + (x(3,k)-x(3,j))**2.)*1.d0
       tmp4 = dsqrt(tmp2)*1.d0
       rpp(1,j,k) = (x(1,k)-x(1,j))/tmp4*1.d0
       rpp(2,j,k) = (x(2,k)-x(2,j))/tmp4*1.d0
       rpp(3,j,k) = (x(3,k)-x(3,j))/tmp4*1.d0
       rpp(1,k,j) = rpp(1,j,k)
       rpp(2,k,j) = rpp(2,j,k)
       rpp(3,k,j) = rpp(3,j,k)
c
       alpha_p(j,k) = 1.17014435E-002/tmp2**1.5*1.d0
       alpha_p(k,j) = alpha_p(j,k)
      end if
```

```
          end do
c
          counter = 0
          do k = 2,nbig
           if (k.ne.j) then
c            tmp1 = r dot s
             tmp1=(rpp(1,j,k)*sx(j)+rpp(2,j,k)*sy(j)+rpp(3,j,
                k)*sz(j))*1.d0
c            tmp3 = s  x  r
             tmp3 = (sy(j)*rpp(3,j,k) - sz(j)*rpp(2,j,k))*1.
                d0
             tmpx = alpha_p(j,k)*m(k)* tmp1 * tmp3
             dsdt(1,j)=(dsdt(1,j) + alpha_p(j,k)*m(k)* tmp1 *
                 tmp3)*1.d0
             tmp3 =(sz(j)*rpp(1,j,k) - sx(j)*rpp(3,j,k))*1.d0
             tmpy = alpha_p(j,k)*m(k)* tmp1 * tmp3
             dsdt(2,j)=(dsdt(2,j) + alpha_p(j,k)*m(k)* tmp1 *
                 tmp3)*1.d0
             tmp3=(sx(j)*rpp(2,j,k) - sy(j)*rpp(1,j,k))*1.d0
             tmpz = alpha_p(j,k)*m(k)* tmp1 * tmp3
             dsdt(3,j)=(dsdt(3,j) + alpha_p(j,k)*m(k)* tmp1 *
                 tmp3)*1.d0
c
           end if
          end do
c
```

```
c
      end do
c
c
      return
      end
c
c
%%%
```

```
c force should not be a function of the velocities.
c
c N.B. All coordinates and velocities must be with respect
     to central body
c ===
c
      ------------------------------------------------------------------


c
      subroutine mfo_user (time,jcen,nbod,nbig,m,x,v,a,s)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig
      real*8 time,jcen(3),m(nbod),x(3,nbod),v(3,nbod),a(3,
     &   nbod)
      real*8 s(3,nbod),dsdt(3,nbod)
c
c Local
      integer i,j,k,ix,mm,nn
      integer :: counter = 0
      real*8 x2,y2,z2,r2,r1,rv,jr2,u2,m_j,r_j,diff
      real*8 tmp1,tmp2,tmp3,atmp(3),atmp_rot(3),tmp,r_p(
     &   NMAX),s_tmp
```

```
      real*8 hx,hy,hz,h2,h,ci,incl(NMAX),node(NMAX)
      real*8 incl_s(NMAX),node_s(NMAX),xtmp,ytmp,ztmp
      real*8 v2,r,temp,e,p,q,aj,dpdt,dqdt,h_len
      real*8 s0,rr,cc,bb,aa,peri,maj,tt,alpha,xyz_tmp(3),
     uvw_tmp(3)
      real*8 dPadt,dpsidt,tmp4,tmp5,ss(3),ll(3),hh(3),yy
     (3),sss(3)
      real*8 dsxdt,dsydt,dszdt,timex,equin_tmp,obl_tmp,
     xyz_rot(3)
      real*8 ls,ll_len,ss_len,incl_tmp,node_tmp,sini,sini2
     ,l1,l2,r3tmp
      real*8 L_orbit(3),arr_tmp1(3),arr_tmp2(3),arr_1(3),
     arr_2(3)
      real*8 P2_tp(9),P3_tp(9),P_tp(9)
      real*8, DIMENSION(NMAX) :: psi = 0.,Pa=0.,sx = 0.,sy
       = 0.,sz = 1.
      real*8, DIMENSION(3, 3) ::P3 = 0.,P2 = 0.,PP = 0.
      real*8, DIMENSION(3, 3) ::LP3_tp = 0.,LP2_tp = 0.,
     LP_tp = 0.
      real*8, DIMENSION(nbig, 3) :: trans,sspin
c
     ------------------------------------------------------------------------


c  spin is updated only once every time step
      r_j = 0.000462392945
```

```
c      scaled mass of jupiter, m(1) is scaled in cgs units
       m_j = 9.5478951698e-04*m(1)
c
c radius of planet in unit of r_j
       do ix = 2, nbig
       tmp = m(ix)/m_j
       r_p(ix) = tmp**.3333333333333333d0
c       r_p(ix) = 2.26574081 * 1.d-5
       end do
c initialize auser
       do j = 1, nbod
            a(1,j) = .0d0
            a(2,j) = .0d0
            a(3,j) = .0d0
       end do
c
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c      calculate solar torque & spin evolution (+e+a)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
       do j=2,nbig
         sx(j) = s(1,j)*1.d0
         sy(j) = s(2,j)*1.d0
         sz(j) = s(3,j)*1.d0
```

```
c          sx(j) = 0.00429949*1.d0
c          sy(j) = 0.00177454*1.d0
c          sz(j) = 0.999989*1.d0
          tmp1 = (sx(j)**2.d0 + sy(j)**2.d0 +sz(j)**2.d0)*1.
             d0
          tmp2 = dsqrt(tmp1)*1.d0
          sx(j) = sx(j)/tmp2*1.d0
          sy(j) = sy(j)/tmp2*1.d0
          sz(j) = sz(j)/tmp2*1.d0
c
         incl_s(j) = acos(sz(j))*1.d0
         if (incl_s(j).gt.1.d-7) then
            node_s(j) = atan2 (sx(j),-sy(j))*1.d0
c             node_s(j) = atan (-sx(j)/sy(j))
         else
            node_s(j) = 0.d0
         end if
          end do
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c
c  rotate force of equator to perpendicular to spin axis
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       do j=2,nbig
```

```
      tmp1 = cos(node_s(j))*1.d0
      tmp2 = sin(node_s(j))*1.d0
      P3(1,1) = tmp1
      P3(1,2) = -tmp2
      P3(2,1) = tmp2
      P3(2,2) = tmp1
      P3(3,3) = 1.
c
      P3_tp(1) = tmp1*1.d0
      P3_tp(2) = tmp2*1.d0
      P3_tp(4) = -tmp2*1.d0
      P3_tp(5) = tmp1*1.d0
      P3_tp(9) = 1.d0
c
c
      tmp1 = cos(incl_s(j))*1.d0
      tmp2 = sin(incl_s(j))*1.d0
      P2(1,1) = 1.d0
      P2(2,2) = tmp1*1.d0
      P2(2,3) = -tmp2*1.d0
      P2(3,2) = tmp2*1.d0
      P2(3,3) = tmp1*1.d0
c
      P2_tp(1) = 1.d0
      P2_tp(5) = tmp1*1.d0
```

```
      P2_tp(6) = tmp2*1.d0
      P2_tp(8) = -tmp2*1.d0
      P2_tp(9) = tmp1*1.d0
c
      PP =MATMUL(P3, P2)
      P_tp=[P3_tp(1)*1.d0,P3_tp(2)*1.d0,0.d0,P2_tp(5)*
     P3_tp(4)*1.d0,
     %  P2_tp(5)*P3_tp(5)*1.d0,P2_tp(6)*1.d0,
     %  P2_tp(8)*P3_tp(4)*1.d0,P2_tp(8)*P3_tp(5)*1.d0,
     P2_tp(9)*1.d0]
c
      do k = 2, nbod
c
        if (k.ne.j) then
        x2 = (x(1,k)-x(1,j))**2
        y2 = (x(2,k)-x(2,j))**2
        z2 = (x(3,k)-x(3,j))**2
        xyz_tmp = [x(1,k)-x(1,j),x(2,k)-x(2,j),x(3,k)-x
     (3,j)]
c        xyz_rot = matmul(P_tp,xyz_tmp)
        xyz_rot(1) = P_tp(1)*xyz_tmp(1)+P_tp(2)*xyz_tmp
     (2)
     %             +P_tp(3)*xyz_tmp(3)
        xyz_rot(2) = P_tp(4)*xyz_tmp(1)+P_tp(5)*xyz_tmp
     (2)
     %             +P_tp(6)*xyz_tmp(3)
```

```
      xyz_rot(3) = P_tp(7)*xyz_tmp(1)+P_tp(8)*xyz_tmp
     (2)
%                       +P_tp(9)*xyz_tmp(3)
      r2 = x2 + y2 + z2
      jr2 = 0.0147*(r_j)**2/r2
      u2 = xyz_rot(3)**2/r2
      tmp1 = m(j)/r2**1.5
      tmp2 = jr2*(7.5d0*u2 - 1.5d0)
      tmp3 = jr2*3.d0
      atmp(1) = xyz_rot(1) * tmp1 * tmp2
      atmp(2) = xyz_rot(2) * tmp1 * tmp2
      atmp(3) = xyz_rot(3) * tmp1 * (tmp2 - tmp3)
      atmp_rot = matmul(PP,atmp)
      a(1,k) =  a(1,k) + atmp_rot(1)
      a(2,k) =  a(2,k) + atmp_rot(2)
      a(3,k) =  a(3,k) + atmp_rot(3)
      end if
c     for k
      end do
c     for j
      end do
c
c
-------------------------------------------------------------------------------
c
```

```
      return
      end
c
c
%%%
```

```
c
c-----------------------------------------------------------------------
c
      subroutine mal_hvar (time,tstart,tstop,dtout,algor,
     &   h0,tol,jcen,
     %   rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh
     &   ,s,rho,rceh,
     %   stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem
     &   ,lmem,onestep,
     %   dsdt)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(nbod),opt(8),opflag,
     &   ngflag,ndump,nfun
      integer lmem(NMESS)
      real*8 time,tstart,tstop,dtout,h0,tol,jcen(3),rcen,
     &   rmax
      real*8 en(3),am(3),cefac,m(nbod),xh(3,nbod),vh(3,
     &   nbod)
      real*8 s(3,nbod),rho(nbod),rceh(nbod),ngf(4,nbod),
     &   dsdt(3,nbod)
      character*8 id(nbod)
```

```
      character*80 outfile(3),dumpfile(4),mem(NMESS)
c
c Local
      integer i,j,k,n,itmp,nhit,ihit(CMAX),jhit(CMAX),chit
     (CMAX)
      integer dtflag,ejflag,nowflag,stopflag,nstored,ce(
     NMAX)
      integer nclo,iclo(CMAX),jclo(CMAX),nce,ice(NMAX),jce
     (NMAX)
      real*8 tmp0,h,hdid,tout,tdump,tfun,tlog,tsmall,
     dtdump,dtfun
      real*8 thit(CMAX),dhit(CMAX),thit1,x0(3,NMAX),v0(3,
     NMAX)
      real*8 rce(NMAX),rphys(NMAX),rcrit(NMAX),a(NMAX)
      real*8 dclo(CMAX),tclo(CMAX),epoch(NMAX)
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX),rtmp,r3tmp,jj
      external mfo_all,onestep
c
c
      ------------------------------------------------------------------------
c
c Initialize variables. DTFLAG = 0 implies first ever call
     to ONESTEP

      dtout = abs(dtout)
```

```
      dtdump = abs(h0) * ndump
      dtfun  = abs(h0) * nfun
      dtflag = 0
      nstored = 0
      tsmall = h0 * 1.d-8
      h = h0
      do j = 2, nbod
        ce(j) = 0.d0
      end do
c
c Calculate close-encounter limits and physical radii for
   massive bodies
      call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,
         nbod,nbig,
     %  m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile
        (2),1)
      do jj = 2, nbig
        r3tmp = m(jj)/9.5478951698e-04*m(1)
        rtmp = r3tmp**.333333333333333d0
c         jr(jj) = rtmp
   *0.0146964*0.000462392945*0.000462392945
      end do

c
c Set up time of next output, times of previous dump, log
   and periodic effect
```

```
      if (opflag.eq.-1) then
        tout = tstart
      else
        n = int (abs (time - tstart) / dtout) + 1
        tout = tstart  +  dtout * sign (dble(n), tstop -
           tstart)
        if ((tstop - tstart)*(tout - tstop).gt.0) tout =
           tstop
      end if
      tdump = time
      tfun  = time
      tlog  = time
c
c
     ------------------------------------------------------------------


c
c  MAIN  LOOP  STARTS  HERE
c
 100  continue
c Is it time for output ?
      if (abs(tout-time).lt.abs(tsmall).and.opflag.ge.-1)
         then
c
c Beware: the integration may change direction at this
   point!!!!
```

```
         if (opflag.eq.-1) dtflag = 0
c
c Output data for all bodies
         call mio_out (time,jcen,rcen,rmax,nbod,nbig,m,xh,
            vh,s,rho,
     %      stat,id,opt,opflag,algor,outfile(1))
         call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,
            stat,id,
     %      0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo
        ,mem,lmem,
     %      outfile,nstored,0)
         tmp0 = tstop - tout
         tout = tout + sign( min( abs(tmp0), abs(dtout) ),
            tmp0 )
c
c Update the data dump files
         do j = 2, nbod
           epoch(j) = time
         end do
         call mio_dump (time,tstart,tstop,dtout,algor,h,tol
            ,jcen,rcen,
     %      rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
        rho,rceh,stat,
     %      id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
         tdump = time
       end if
```

```
c
c If integration has finished return to the main part of
   programme
      if (abs(tstop-time).le.abs(tsmall).and.opflag.ne.-1)
         return
c
c Set the timestep
      if (opflag.eq.-1) tmp0 = tstart - time
      if (opflag.eq.-2) tmp0 = tstop  - time
      if (opflag.ge.0)  tmp0 = tout   - time
      h = sign ( max( min( abs(tmp0), abs(h) ), tsmall),
         tmp0 )
c
c Save the current coordinates and velocities
      call mco_iden (time,jcen,nbod,nbig,h,m,xh,vh,x0,v0,
         ngf,ngflag,opt)
c
c Advance one timestep
      call onestep (time,h,hdid,tol,jcen,nbod,nbig,m,xh,vh
         ,s,rphys,
     %  rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce,
        mfo_all,dsdt)
      time = time + hdid
c
c Check if close encounters or collisions occurred
      nclo = 0
```

```
      call mce_stat (time,h,rcen,nbod,nbig,m,x0,v0,xh,vh,
          rce,rphys,
     %  nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,ihit,
        jhit,
     %  chit,dhit,thit,thit1,nowflag,stat,outfile(3),mem,
        lmem)
c
c
      ------------------------------------------------------------------

c
c  CLOSE ENCOUNTERS
c
c If encounter minima occurred, output details and decide
   whether to stop
       if (nclo.gt.0.and.opflag.ge.-1) then
         itmp = 1
         if (nhit.ne.0) itmp = 0
         call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,
             stat,id,nclo,
     %      iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,
        mem,lmem,
     %      outfile,nstored,itmp)
         if (stopflag.eq.1) return
       end if
c
```

```
c
c-----------------------------------------------------------------------
c
c  COLLISIONS
c
c If a collision occurred, output details and resolve the
   collision
      if (nhit.gt.0.and.opt(2).ne.0) then
        do k = 1, nhit
          if (chit(k).eq.1) then
            i = ihit(k)
            j = jhit(k)
            call mce_coll (thit(k),tstart,en(3),jcen,i,j,
               nbod,nbig,m,xh,
     %         vh,s,rphys,stat,id,opt,mem,lmem,outfile(3))
          end if
        end do
c
c Remove lost objects, reset flags and recompute Hill and
   physical radii
        call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,
           ngf,stat,
     %    id,mem,lmem,outfile(3),itmp)
        dtflag = 1
        if (opflag.ge.0) opflag = 1
```

```
         call mce_init (tstart,algor,h0,jcen,rcen,rmax,
            cefac,nbod,nbig,
     %     m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,
           outfile(2),1)
        end if
c
c
c-----------------------------------------------------------------------

c
c  COLLISIONS WITH CENTRAL BODY
c
c Check for collisions
        call mce_cent (time,hdid,rcen,jcen,2,nbod,nbig,m,x0,
           v0,xh,vh,nhit,
     %  jhit,thit,dhit,algor,ngf,ngflag)
c
c Resolve the collisions
        if (nhit.gt.0) then
          do k = 1, nhit
            i = 1
            j = jhit(k)
            call mce_coll (thit(k),tstart,en(3),jcen,i,j,
               nbod,nbig,m,xh,
     %        vh,s,rphys,stat,id,opt,mem,lmem,outfile(3))
          end do
```

```
c
c Remove lost objects, reset flags and recompute Hill and
   physical radii
        call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,
           ngf,stat,
     %     id,mem,lmem,outfile(3),itmp)
        dtflag = 1
        if (opflag.ge.0) opflag = 1
        call mce_init (tstart,algor,h0,jcen,rcen,rmax,
           cefac,nbod,nbig,
     %     m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,
        outfile(2),0)
      end if
c
c
   ------------------------------------------------------------------------


c
c  DATA DUMP AND PROGRESS REPORT
c
c Do the data dump
      if (abs(time-tdump).ge.abs(dtdump).and.opflag.ge.-1)
         then
        do j = 2, nbod
          epoch(j) = time
        end do
```

```
            call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,
                stat,id,
     %      0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo
            ,mem,lmem,
     %      outfile,nstored,0)
            call mio_dump (time,tstart,tstop,dtout,algor,h,tol
                ,jcen,rcen,
     %      rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
            rho,rceh,stat,
     %      id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
            tdump = time
          end if
c
c Write a progress report to the log file
          if (abs(time-tlog).ge.abs(dtdump).and.opflag.ge.0)
             then
            call mxx_en (jcen,nbod,nbig,m,xh,vh,s,en(2),am(2))
            call mio_log (time,tstart,en,am,opt,mem,lmem)
            tlog = time
          end if
c
c
      ------------------------------------------------------------------------
c
c  CHECK FOR EJECTIONS AND DO OTHER PERIODIC
```

```
   EFFECTS
c
      if (abs(time-tfun).ge.abs(dtfun).and.opflag.ge.-1)
         then
c
c Recompute close encounter limits, to allow for changes
   in Hill radii
         call mce_hill (nbod,m,xh,vh,rce,a)
         do j = 2, nbod
           rce(j) = rce(j) * rceh(j)
         end do
c
c Check for ejections
         call mxx_ejec (time,tstart,rmax,en,am,jcen,2,nbod,
            nbig,m,xh,vh,
     %      s,stat,id,opt,ejflag,outfile(3),mem,lmem)
c
c Remove lost objects, reset flags and recompute Hill and
   physical radii
         if (ejflag.ne.0) then
           call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,
              rcrit,ngf,stat,
     %        id,mem,lmem,outfile(3),itmp)
           dtflag = 1
           if (opflag.ge.0) opflag = 1
           call mce_init (tstart,algor,h0,jcen,rcen,rmax,
```

```
                cefac,nbod,nbig,
     %          m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,
        outfile(2),0)
        end if
        tfun = time
      end if
c
c Go on to the next time step
      goto 100
c
c
c ------------------------------------------------------------------------------

c
      end
c
c
c %%%%
```

```
c Author: John E. Chambers / Yu-Cian Hong
c
c Given initial and final coordinates and velocities, the
   routine returns
c the X and Y coordinates of a box bounding the motion in
   between the
c end points.
c
c If the X or Y velocity changes sign, the routine
   performs a quadratic
c interpolation to estimate the corresponding extreme
   value of X or Y.
c
c
   ------------------------------------------------------------------------


c
      subroutine mce_box (nbod,h,x0,v0,x1,v1,xmin,xmax,
     &   ymin,ymax)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod
      real*8 h,x0(3,nbod), x1(3,nbod), v0(3,nbod),v1(3,
```

```
          nbod)
        real*8    xmin(nbod), xmax(nbod), ymin(nbod),ymax(
          nbod)
c
c Local
        integer j
        real*8 temp
c
c
    ------------------------------------------------------------------------


c
        do j = 2, nbod
          xmin(j) = min (x0(1,j), x1(1,j))
          xmax(j) = max (x0(1,j), x1(1,j))
          ymin(j) = min (x0(2,j), x1(2,j))
          ymax(j) = max (x0(2,j), x1(2,j))
c
c If velocity changes sign, do an interpolation
          if ((v0(1,j).lt.0.and.v1(1,j).gt.0).or.
     %       (v0(1,j).gt.0.and.v1(1,j).lt.0)) then
            temp = (v0(1,j)*x1(1,j) - v1(1,j)*x0(1,j)
     %              - .5d0*h*v0(1,j)*v1(1,j)) / (v0(1,j) -
          v1(1,j))
            xmin(j) = min (xmin(j),temp)
            xmax(j) = max (xmax(j),temp)
```

```
          end if
c
          if ((v0(2,j).lt.0.and.v1(2,j).gt.0).or.
     %        (v0(2,j).gt.0.and.v1(2,j).lt.0)) then
            temp = (v0(2,j)*x1(2,j) - v1(2,j)*x0(2,j)
     %              - .5d0*h*v0(2,j)*v1(2,j)) / (v0(2,j) -
     v1(2,j))
            ymin(j) = min (ymin(j),temp)
            ymax(j) = max (ymax(j),temp)
          end if
        end do
c
c
c-----------------------------------------------------------------------
c
        return
        end
c
c
c%%%
```

```
%%%
```

```
c
c Input/Output
      integer i0, nbod, nbig, nhit, jhit(CMAX), algor,
     ngflag
      real*8 time,h,rcen,jcen(3),m(nbod),x0(3,nbod),v0(3,
     nbod)
      real*8 x1(3,nbod),v1(3,nbod),thit(CMAX),dhit(CMAX),
     ngf(4,nbod)
c
c Local
      integer j
      real*8 rcen2,mco_acsh,a,q,u0,uhit,m0,mhit,mm,r0,mcen
      real*8 hx,hy,hz,h2,p,rr0,rr1,rv0,rv1,temp,e,v2,
     rr_min,vej,r2max
      real*8 xu0(3,NMAX),xu1(3,NMAX),vu0(3,NMAX),vu1(3,
     NMAX)
c
c
     ------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
      nhit = 0
      rcen2 = rcen * rcen
      mcen = m(1)
c
```

```
c If using close-binary code, convert to coords with
   respect to the binary
c       if (algor.eq.11) then
c         mcen = m(1) + m(2)
c         call mco_h2ub (temp,jcen,nbod,nbig,h,m,x0,v0,xu0,
   vu0,ngf,ngflag)
c         call mco_h2ub (temp,jcen,nbod,nbig,h,m,x1,v1,xu1,
   vu1,ngf,ngflag)
c       end if
c
c Distance of closest massive body to central body
      rr_min = 9.9d99
      do j = i0, nbig
        if (algor.eq.11) then
          rr0 = xu0(1,j)*xu0(1,j) + xu0(2,j)*xu0(2,j) +xu0
             (3,j)*xu0(3,j)
        else
          rr0 = x0(1,j)*x0(1,j) + x0(2,j)*x0(2,j) + x0(3,j
             )*x0(3,j)
        end if
        rr_min = min(rr_min, rr0)
      end do
c
c Maximum plausible distance from central body compatible
   with collision
      vej = dsqrt(2.d0 * mcen / rcen)
```

```
      temp = rcen  +  0.5d0 * h * vej
      r2max = temp * temp
c
c Check for collisions with the central body
      do j = i0, nbod
        if (algor.eq.11) then
          rr0 = xu0(1,j)*xu0(1,j) + xu0(2,j)*xu0(2,j) +xu0
     (3,j)*xu0(3,j)
          rr1 = xu1(1,j)*xu1(1,j) + xu1(2,j)*xu1(2,j) +xu1
     (3,j)*xu1(3,j)
          rv0 = vu0(1,j)*xu0(1,j) + vu0(2,j)*xu0(2,j) +vu0
     (3,j)*xu0(3,j)
          rv1 = vu1(1,j)*xu1(1,j) + vu1(2,j)*xu1(2,j) +vu1
     (3,j)*xu1(3,j)
        else
          rr0 = x0(1,j)*x0(1,j) + x0(2,j)*x0(2,j) + x0(3,j
     )*x0(3,j)
          rr1 = x1(1,j)*x1(1,j) + x1(2,j)*x1(2,j) + x1(3,j
     )*x1(3,j)
          rv0 = v0(1,j)*x0(1,j) + v0(2,j)*x0(2,j) + v0(3,j
     )*x0(3,j)
          rv1 = v1(1,j)*x1(1,j) + v1(2,j)*x1(2,j) + v1(3,j
     )*x1(3,j)
        end if
c
c If inside the central body, or passing through
```

```
   pericentre, use 2-body approx.
         if ((rv0*h.le.0.and.rv1*h.ge.0).or.min(rr0,rr1).le
            .rcen2) then
c
c If outside other massive bodies, is an artifact due to
   barycentric motion
c           if (rr0.gt.rr_min) goto 100
c
c Ignore if too far from central body to plausibly have
   had a collision
           if (min(rr0,rr1).gt.r2max) goto 100
c
           if (algor.eq.11) then
             hx = xu0(2,j) * vu0(3,j) - xu0(3,j) * vu0(2,
                j)
             hy = xu0(3,j) * vu0(1,j) - xu0(1,j) * vu0(3,
                j)
             hz = xu0(1,j) * vu0(2,j) - xu0(2,j) * vu0(1,
                j)
             v2 = vu0(1,j)*vu0(1,j) +vu0(2,j)*vu0(2,j) +vu0
                (3,j)*vu0(3,j)
           else
             hx = x0(2,j) * v0(3,j) - x0(3,j) * v0(2,j)
             hy = x0(3,j) * v0(1,j) - x0(1,j) * v0(3,j)
             hz = x0(1,j) * v0(2,j) - x0(2,j) * v0(1,j)
             v2 = v0(1,j)*v0(1,j) + v0(2,j)*v0(2,j) + v0(3,
```

```
                j)*v0(3,j)
            end if
            h2 = hx*hx + hy*hy + hz*hz
            p = h2 / (mcen + m(j))
            r0 = dsqrt(rr0)
            temp = 1.d0 + p*(v2/(mcen + m(j)) - 2.d0/r0)
            e = dsqrt( max(temp,0.d0) )
            q = p / (1.d0 + e)
c
c If the object hit the central body
            if (q.le.rcen) then
              nhit = nhit + 1
              jhit(nhit) = j
              dhit(nhit) = rcen
c
c Time of impact relative to the end of the timestep
              if (e.lt.1) then
                a = q / (1.d0 - e)
                uhit = sign (acos((1.d0 - rcen/a)/e), -h)
                u0   = sign (acos((1.d0 - r0/a  )/e), rv0)
                mhit = mod (uhit - e*sin(uhit) + PI, TWOPI)
                   - PI
                m0   = mod (u0   - e*sin(u0)   + PI, TWOPI)
                   - PI
              else
                a = q / (e - 1.d0)
```

```
                uhit = sign (mco_acsh((1.d0 - rcen/a)/e), -h
                   )
                u0   = sign (mco_acsh((1.d0 - r0/a  )/e),
                   rv0)
                mhit = mod (uhit - e*sinh(uhit) + PI, TWOPI)
                    - PI
                m0   = mod (u0   - e*sinh(u0)   + PI, TWOPI)
                    - PI
              end if
              mm = dsqrt((mcen + m(j)) / (a*a*a))
              thit(nhit) = (mhit - m0) / mm + time
            end if
100         continue
          end if
        end do
c
c
c-----------------------------------------------------------------------
c
        return
        end
c
c%%%
```

```
c        MCE_COLL.FOR    (May 2019)
c
c
%%%
```

```
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i,j,nbod,nbig,stat(nbod),opt(8),lmem(NMESS)
      real*8 time,tstart,elost,jcen(3)
      real*8 m(nbod),xh(3,nbod),vh(3,nbod),s(3,nbod),rphys
     (nbod)
      character*80 outfile,mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer year,month,itmp
      real*8 t1
      character*38 flost,fcol
      character*6 tstring
c
c
     ------------------------------------------------------------------------


c
c If two bodies collided, check that the less massive one
    is removed
c (unless the more massive one is a Small body)
      if (i.gt.1) then
```

```
          if (m(j).gt.m(i).and.j.le.nbig) then
            itmp = i
            i = j
            j = itmp
          end if
        end if
c
c Write message to info file (I=0 implies collision with
   the central body)
  10  open (23, file=outfile, status='old', access='append
     ', err=10)
c
        if (opt(3).eq.1) then
          call mio_jd2y (time,year,month,t1)
          if (i.eq.0.or.i.eq.1) then
            flost = '(1x,a8,a,i10,1x,i2,1x,f8.5)'
            write (23,flost) id(j),mem(67)(1:lmem(67)),year,
               month,t1
          else
            fcol  = '(1x,a8,a,a8,a,i10,1x,i2,1x,f4.1)'
            write (23,fcol) id(i),mem(69)(1:lmem(69)),id(j),
     %        mem(71)(1:lmem(71)),year,month,t1
          end if
        else
          if (opt(3).eq.3) then
            t1 = (time - tstart) / 365.25d0
```

```
            tstring = mem(2)
            flost = '(1x,a8,a,f18.7,a)'
            fcol  = '(1x,a8,a,a8,a,1x,f14.3,a)'
          else
            if (opt(3).eq.0) t1 = time
            if (opt(3).eq.2) t1 = time - tstart
            tstring = mem(1)
            flost = '(1x,a8,a,f18.5,a)'
            fcol  = '(1x,a8,a,a8,a,1x,f14.1,a)'
          end if
          if (i.eq.0.or.i.eq.1) then
            write (23,flost) id(j),mem(67)(1:lmem(67)),t1,
               tstring
          else
            write (23,fcol) id(i),mem(69)(1:lmem(69)),id(j),
     %       mem(71)(1:lmem(71)),t1,tstring
          end if
        end if
        close (23)
c
c Do the collision (inelastic merger)
        call mce_merg (jcen,i,j,nbod,nbig,m,xh,vh,s,stat,
           elost)
c
c
```

```
c
      return
      end
c
c
%%%
```

```
c where R is the current distance from the central body.
c
c The routine also gives the semi-major axis, A, of each
   object's orbit.
c
c N.B. Designed to use heliocentric coordinates, but
   should be adequate using
c ===  barycentric coordinates.
c
c
   ------------------------------------------------------------------------


c
      subroutine mce_hill (nbod,m,x,v,hill,a)
c
      implicit none
      include 'mercury.inc'
      real*8 THIRD
      parameter (THIRD = .3333333333333333d0)
c
c Input/Output
      integer nbod
      real*8 m(nbod),x(3,nbod),v(3,nbod),hill(nbod),a(nbod
         )
c
c Local
```

```
      integer j
      real*8 r, v2, gm
c
c
c-----------------------------------------------------------------------

c
      do j = 2, nbod
        gm = m(1) + m(j)
         call mco_x2a (gm,x(1,j),x(2,j),x(3,j),v(1,j),v(2,j
     %      ),v(3,j),a(j),
     %      r,v2)
c If orbit is hyperbolic, use the distance rather than the
     % semi-major axis
         if (a(j).le.0) a(j) = r
         hill(j) = a(j) * (THIRD * m(j) / m(1))**THIRD
      end do
c
c
c-----------------------------------------------------------------------

c
      return
      end
c
c
```

```
%%%
```

```
c section 4.2 of Chambers / Yu-Cian Hong 1999, Monthly
   Notices, vol 304, p793-799).
c
c N.B. Designed to use heliocentric coordinates, but
   should be adequate using
c ===  barycentric coordinates.
c
c
   ------------------------------------------------------------------


c
      subroutine mce_init (tstart,algor,h,jcen,rcen,rmax,
         cefac,nbod,
     %  nbig,m,x,v,s,rho,rceh,rphys,rce,rcrit,id,opt,
        outfile,rcritflag)
c
      implicit none
      include 'mercury.inc'
c
      real*8 N2,THIRD
      parameter (N2=.4d0,THIRD=.3333333333333333d0)
c
c Input/Output
      integer nbod,nbig,algor,opt(8),rcritflag
      real*8 tstart,h,jcen(3),rcen,rmax,cefac,m(nbod),x(3,
         nbod)
```

```
      real*8 v(3,nbod),s(3,nbod),rho(nbod),rceh(nbod),
     &   rphys(nbod)
      real*8 rce(nbod),rcrit(nbod)
      character*8 id(nbod)
      character*80 outfile
c
c Local
      integer j
      real*8 a(NMAX),hill(NMAX),temp,amin,vmax,k_2,rhocgs,
     &   rcen_2
      character*80 header,c(NMAX)
      character*8 mio_re2c, mio_fl2c
c
c
     ------------------------------------------------------------------------


c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
      amin = HUGE
c
c Calculate the Hill radii
      call mce_hill (nbod,m,x,v,hill,a)
c
c Determine the maximum close-encounter distances, and the
```

```
     physical radii
       temp = 2.25d0 * m(1) / PI
       do j = 2, nbod
         rce(j)   = hill(j) * rceh(j)
         rphys(j) = hill(j) / a(j) * (temp/rho(j))**THIRD
         amin = min (a(j), amin)
       end do
c
c If required, calculate the changeover distance used by
   hybrid algorithm
       if (rcritflag.eq.1) then
         vmax = dsqrt (m(1) / amin)
         temp = N2 * h * vmax
         do j = 2, nbod
           rcrit(j) = max(hill(j) * cefac, temp)
         end do
       end if
c
c Write list of object's identities to close-encounter
   output file
       header(1:8)   = mio_fl2c (tstart)
       header(9:16)  = mio_re2c (dble(nbig - 1),   0.d0,
          11239423.99d0)
       header(12:19) = mio_re2c (dble(nbod - nbig),0.d0,
          11239423.99d0)
       header(15:22) = mio_fl2c (m(1) * k_2)
```

```
      header(23:30) = mio_fl2c (jcen(1) * rcen_2)
      header(31:38) = mio_fl2c (jcen(2) * rcen_2 * rcen_2)
      header(39:46) = mio_fl2c (jcen(3) * rcen_2 * rcen_2
     *  * rcen_2)
      header(47:54) = mio_fl2c (rcen)
      header(55:62) = mio_fl2c (rmax)
c
      do j = 2, nbig
        c(j)(1:8) = mio_re2c (dble(j - 1), 0.d0,
     *    11239423.99d0)
        c(j)(4:11) = id(j)
        c(j)(12:19) = mio_fl2c (m(j) * k_2)
c        c(j)(20:27) = mio_fl2c (s(1,j) * k_2)
c        c(j)(28:35) = mio_fl2c (s(2,j) * k_2)
c        c(j)(36:43) = mio_fl2c (s(3,j) * k_2)
        c(j)(20:27) = mio_fl2c (s(1,j))
        c(j)(28:35) = mio_fl2c (s(2,j))
        c(j)(36:43) = mio_fl2c (s(3,j))
        c(j)(44:51) = mio_fl2c (rho(j) / rhocgs)
      end do
c
c Write compressed output to file
  50  open (22, file=outfile, status='old', access='append
     *', err=50)
      write (22,'(a1,a2,i2,a62,i1)') char(12),'6a',algor,
     *  header(1:62),
```

```
      %  opt(4)
       do j = 2, nbig
         write (22,'(a51)') c(j)(1:51)
       end do
       close (22)
c
c
      ------------------------------------------------------------------

c
       return
       end
c
c
      %%%%
```

```
c conserving mass and linear momentum.
c    If J <= NBIG, then J is a Big body
c    If J >  NBIG, then J is a Small body
c    If I = 0, then I is the central body
c
c N.B. All coordinates and velocities must be with respect
     to central body.
c ===
c
c
```

```
c
      subroutine mce_merg (jcen,i,j,nbod,nbig,m,xh,vh,s,
     &   stat,elost)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i, j, nbod, nbig, stat(nbod)
      real*8 jcen(3),m(nbod),xh(3,nbod),vh(3,nbod),s(3,
     &   nbod),elost
c
c Local
      integer k
```

```
      real*8 tmp1, tmp2, dx, dy, dz, du, dv, dw, msum,
     &   mredu, msum_1
      real*8 e0, e1, l2
c
c
     ------------------------------------------------------------------


c
c If a body hits the central body
      if (i.le.1) then
        call mxx_en (jcen,nbod,nbig,m,xh,vh,s,e0,l2)
c
c If a body hit the central body...
        msum    = m(1) + m(j)
        msum_1 = 1.d0 / msum
        mredu   = m(1) * m(j) * msum_1
        dx = xh(1,j)
        dy = xh(2,j)
        dz = xh(3,j)
        du = vh(1,j)
        dv = vh(2,j)
        dw = vh(3,j)
c
c Calculate new spin angular momentum of the central body
C         s(1,1) = s(1,1) + s(1,j) + mredu * (dy * dw
     &  - dz * dv)
```

```
C          s(2,1) = s(2,1)  +  s(2,j)  +  mredu * (dz * du
     -  dx * dw)
C          s(3,1) = s(3,1)  +  s(3,j)  +  mredu * (dx * dv
     -  dy * du)
           s(1,1) = s(1,1)    +  mredu * (dy * dw  -  dz * dv
              )
           s(2,1) = s(2,1)    +  mredu * (dz * du  -  dx * dw
              )
           s(3,1) = s(3,1)    +  mredu * (dx * dv  -  dy * du
              )
c
c Calculate shift in barycentric coords and velocities of
     central body
           tmp2 = m(j) * msum_1
           xh(1,1) = tmp2 * xh(1,j)
           xh(2,1) = tmp2 * xh(2,j)
           xh(3,1) = tmp2 * xh(3,j)
           vh(1,1) = tmp2 * vh(1,j)
           vh(2,1) = tmp2 * vh(2,j)
           vh(3,1) = tmp2 * vh(3,j)
           m(1) = msum
           m(j) = 0.d0
           s(1,j) = 0.d0
           s(2,j) = 0.d0
           s(3,j) = 0.d0
c
```

```
c Shift the heliocentric coordinates and velocities of all
    bodies
          do k = 2, nbod
            xh(1,k) = xh(1,k) - xh(1,1)
            xh(2,k) = xh(2,k) - xh(2,1)
            xh(3,k) = xh(3,k) - xh(3,1)
            vh(1,k) = vh(1,k) - vh(1,1)
            vh(2,k) = vh(2,k) - vh(2,1)
            vh(3,k) = vh(3,k) - vh(3,1)
          end do
c
c Calculate energy loss due to the collision
          call mxx_en (jcen,nbod,nbig,m,xh,vh,s,e1,l2)
          elost = elost + (e0 - e1)
        else
c
c If two bodies collided...
          msum    = m(i) + m(j)
          msum_1 = 1.d0 / msum
          mredu   = m(i) * m(j) * msum_1
          dx = xh(1,i) - xh(1,j)
          dy = xh(2,i) - xh(2,j)
          dz = xh(3,i) - xh(3,j)
          du = vh(1,i) - vh(1,j)
          dv = vh(2,i) - vh(2,j)
          dw = vh(3,i) - vh(3,j)
```

```
c
c Calculate energy loss due to the collision
        elost = elost  +  .5d0 * mredu * (du*du + dv*dv +
           dw*dw)
     %        -  m(i) * m(j) / dsqrt(dx*dx + dy*dy + dz*dz
          )
c
c Calculate spin angular momentum of the new body
C         s(1,i) = s(1,i)  +  s(1,j)  +  mredu * (dy * dw
   -  dz * dv)
C         s(2,i) = s(2,i)  +  s(2,j)  +  mredu * (dz * du
   -  dx * dw)
C         s(3,i) = s(3,i)  +  s(3,j)  +  mredu * (dx * dv
   -  dy * du)
          s(1,i) = s(1,i)
          s(2,i) = s(2,i)
          s(3,i) = s(3,i)

c
c Calculate new coords and velocities by conserving centre
     of mass & momentum
          tmp1 = m(i) * msum_1
          tmp2 = m(j) * msum_1
          xh(1,i) = xh(1,i) * tmp1  +  xh(1,j) * tmp2
          xh(2,i) = xh(2,i) * tmp1  +  xh(2,j) * tmp2
          xh(3,i) = xh(3,i) * tmp1  +  xh(3,j) * tmp2
```

```
            vh(1,i) = vh(1,i) * tmp1  +  vh(1,j) * tmp2
            vh(2,i) = vh(2,i) * tmp1  +  vh(2,j) * tmp2
            vh(3,i) = vh(3,i) * tmp1  +  vh(3,j) * tmp2
            m(i) = msum
          end if
c
c Flag the lost body for removal, and move it away from
   the new body
          stat(j) = -2
          xh(1,j) = -xh(1,j)
          xh(2,j) = -xh(2,j)
          xh(3,j) = -xh(3,j)
          vh(1,j) = -vh(1,j)
          vh(2,j) = -vh(2,j)
          vh(3,j) = -vh(3,j)
          m(j)    = 0.d0
          s(1,j) = 0.d0
          s(2,j) = 0.d0
          s(3,j) = 0.d0
c
c
      ------------------------------------------------------------------
c
          return
          end
```

```
c
c
%%%
```

```
c
     ------------------------------------------------------------------

c
      subroutine mce_min (d0,d1,d0t,d1t,h,d2min,tmin)
c
      implicit none
c
c Input/Output
      real*8 d0,d1,d0t,d1t,h,d2min,tmin
c
c Local
      real*8 a,b,c,temp,tau
c
c
     ------------------------------------------------------------------

c
      if (d0t*h.gt.0.or.d1t*h.lt.0) then
        if (d0.le.d1) then
          d2min = d0
          tmin = -h
        else
          d2min = d1
          tmin = 0.d0
        end if
```

```
      else
        temp = 6.d0*(d0 - d1)
        a = temp + 3.d0*h*(d0t + d1t)
        b = temp + 2.d0*h*(d0t + 2.d0*d1t)
        c = h * d1t
c
        temp =-.5d0*(b + sign (dsqrt(max(b*b - 4.d0*a*c,0.
           d0)), b) )
        if (temp.eq.0) then
          tau = 0.d0
        else
          tau = c / temp
        end if
c
c Make sure TAU falls in the interval -1 < TAU < 0
        tau = min(tau, 0.d0)
        tau = max(tau, -1.d0)
c
c Calculate TMIN and D2MIN
        tmin = tau * h
        temp = 1.d0 + tau
        d2min = tau*tau*((3.d0+2.d0*tau)*d0 + temp*h*d0t)
     %        + temp*temp*((1.d0-2.d0*tau)*d1 + tau*h*d1t)
c
c Make sure D2MIN is not negative
        d2min = max(d2min, 0.d0)
```

```
      end if
c
c
c     ------------------------------------------------------------------

c
      return
      end
c
c
c
c     %%%%
```

```
c and final coordinates X(0),X(1) and velocities V(0),V(1)
     of the step, and
c the stepsize H.
c
c  ICLO, JCLO contain the indices of the objects
c  DCLO is their minimum separation
c  TCLO is the time of closest approach relative to
     current time
c
c The routine also checks for collisions/near misses given
     the physical radii
c RPHYS, and returns the time THIT of the collision/near
     miss closest to the
c start of the timestep, and the identities IHIT and JHIT
     of the objects
c involved.
c
c  NHIT = +1 implies a collision
c         -1    "    a near miss
c
c N.B. All coordinates & velocities must be with respect
     to the central body!!
c ===
c
      ------------------------------------------------------------------
```

```
c
      subroutine mce_stat (time,h,rcen,nbod,nbig,m,x0,v0,
     &   x1,v1,rce,
     %  rphys,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,
     &   ihit,jhit,
     %  chit,dhit,thit,thit1,nowflag,stat,outfile,mem,lmem
     &   )
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,stat(nbod),nowflag
      integer nclo,iclo(CMAX),jclo(CMAX)
      integer nhit,ihit(CMAX),jhit(CMAX),chit(CMAX),lmem(
     &   NMESS)
      real*8 time,h,rcen,m(nbod),x0(3,nbod),v0(3,nbod)
      real*8 x1(3,nbod),v1(3,nbod),rce(nbod),rphys(nbod)
      real*8 dclo(CMAX),tclo(CMAX),thit(CMAX),dhit(CMAX),
     &   thit1
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX)
      character*80 outfile,mem(NMESS)
c
c Local
      integer i,j
      real*8 d0,d1,d0t,d1t,hm1,tmp0,tmp1
```

```
      real*8 dx0,dy0,dz0,du0,dv0,dw0,dx1,dy1,dz1,du1,dv1,
     &   dw1
      real*8 xmin(NMAX),xmax(NMAX),ymin(NMAX),ymax(NMAX)
      real*8 d2min,d2ce,d2near,d2hit,temp,tmin
c
c
      ------------------------------------------------------------------------


c
      nhit = 0
      thit1 = sign(HUGE, h)
      hm1 = 1.d0 / h
c
c Calculate maximum and minimum values of x and y coords
     for each object
      call mce_box (nbod,h,x0,v0,x1,v1,xmin,xmax,ymin,ymax
     &   )
c
c Adjust values by the maximum close-encounter radius plus
     a fudge factor
      do j = 2, nbod
        temp = rce(j) * 1.2d0
        xmin(j) = xmin(j) - temp
        xmax(j) = xmax(j) + temp
        ymin(j) = ymin(j) - temp
        ymax(j) = ymax(j) + temp
```

```
      end do
c
c Check for close encounters between each pair of objects
      do i = 2, nbig
        do j = i + 1, nbod
          if (    xmax(i).ge.xmin(j).and.xmax(j).ge.xmin(i)
     %       .and.ymax(i).ge.ymin(j).and.ymax(j).ge.ymin(i)
     %       .and.stat(i).ge.0.and.stat(j).ge.0) then
c
c If the X–Y boxes for this pair overlap, check
   circumstances more closely
            dx0 = x0(1,i) - x0(1,j)
            dy0 = x0(2,i) - x0(2,j)
            dz0 = x0(3,i) - x0(3,j)
            du0 = v0(1,i) - v0(1,j)
            dv0 = v0(2,i) - v0(2,j)
            dw0 = v0(3,i) - v0(3,j)
            d0t = (dx0*du0 + dy0*dv0 + dz0*dw0) * 2.d0
c
            dx1 = x1(1,i) - x1(1,j)
            dy1 = x1(2,i) - x1(2,j)
            dz1 = x1(3,i) - x1(3,j)
            du1 = v1(1,i) - v1(1,j)
            dv1 = v1(2,i) - v1(2,j)
            dw1 = v1(3,i) - v1(3,j)
            d1t = (dx1*du1 + dy1*dv1 + dz1*dw1) * 2.d0
```

```
c
c Estimate minimum separation during the time interval,
   using interpolation
              d0 = dx0*dx0 + dy0*dy0 + dz0*dz0
              d1 = dx1*dx1 + dy1*dy1 + dz1*dz1
              call mce_min (d0,d1,d0t,d1t,h,d2min,tmin)
              d2ce  = max (rce(i), rce(j))
              d2hit = rphys(i) + rphys(j)
              d2ce   = d2ce  * d2ce
              d2hit  = d2hit * d2hit
              d2near = d2hit * 4.d0
c
c If the minimum separation qualifies as an encounter or
   if a collision
c is in progress, store details
              if((d2min.le.d2ce.and.d0t*h.le.0.and.d1t*h.ge
                 .0
     %          .and.j.le.nbig).or.(d2min.le.d2hit.and.j.le.
        nbig)) then
                nclo = nclo + 1
                if (nclo.gt.CMAX) then
 230              open (23,file=outfile,status='old',access
    ='append',
     %              err=230)
                  write (23,'(/,2a,/,a)') mem(121)(1:lmem
                     (121)),
```

```
      %              mem(132)(1:lmem(132)),mem(82)(1:lmem(82)
          )
                     close (23)
                   else
                     tclo(nclo) = tmin + time
                     dclo(nclo) = dsqrt (max(0.d0,d2min))
                     iclo(nclo) = i
                     jclo(nclo) = j
c
c Make sure the more massive body is listed first
                     if (m(j).gt.m(i).and.j.le.nbig) then
                       iclo(nclo) = j
                       jclo(nclo) = i
                     end if
c
c Make linear interpolation to get coordinates at time of
    closest approach
                     tmp0 = 1.d0 + tmin*hm1
                     tmp1 = -tmin*hm1
                     ixvclo(1,nclo) = tmp0 * x0(1,i)  +  tmp1 *
                         x1(1,i)
                     ixvclo(2,nclo) = tmp0 * x0(2,i)  +  tmp1 *
                         x1(2,i)
                     ixvclo(3,nclo) = tmp0 * x0(3,i)  +  tmp1 *
                         x1(3,i)
                     ixvclo(4,nclo) = tmp0 * v0(1,i)  +  tmp1 *
```

```
                v1(1,i)
              ixvclo(5,nclo) = tmp0 * v0(2,i)  +  tmp1 *
                v1(2,i)
              ixvclo(6,nclo) = tmp0 * v0(3,i)  +  tmp1 *
                v1(3,i)
              jxvclo(1,nclo) = tmp0 * x0(1,j)  +  tmp1 *
                x1(1,j)
              jxvclo(2,nclo) = tmp0 * x0(2,j)  +  tmp1 *
                x1(2,j)
              jxvclo(3,nclo) = tmp0 * x0(3,j)  +  tmp1 *
                x1(3,j)
              jxvclo(4,nclo) = tmp0 * v0(1,j)  +  tmp1 *
                v1(1,j)
              jxvclo(5,nclo) = tmp0 * v0(2,j)  +  tmp1 *
                v1(2,j)
              jxvclo(6,nclo) = tmp0 * v0(3,j)  +  tmp1 *
                v1(3,j)
            end if
          end if
c
c Check for a near miss or collision
          if (d2min.le.d2hit) then
            nhit = nhit + 1
            ihit(nhit) = i
            jhit(nhit) = j
            thit(nhit) = tmin + time
```

```
                dhit(nhit) = dsqrt(d2min)
                chit(nhit) = 1
c
c Make sure the more massive body is listed first
                if (m(jhit(nhit)).gt.m(ihit(nhit)).and.j.le.
                   nbig) then
                  ihit(nhit) = j
                  jhit(nhit) = i
                end if
c
c Is this the collision closest to the start of the time
   step?
                if ((tmin-thit1)*h.lt.0) then
                  thit1 = tmin
                  nowflag = 0
                  if (d1.le.d2hit) nowflag = 1
                end if
              end if
            end if
c
c Move on to the next pair of objects
          end do
        end do
c
c
```

```
c
      return
      end
c
c
%%%
```

```
     v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),
     x(3,nbod)
      real*8 v(3,nbod),ngf(4,nbod)
c
c Local
      integer j
      real*8 mtot,temp
c
c
c-----------------------------------------------------------------------
c
      mtot = 0.d0
      x(1,1) = 0.d0
      x(2,1) = 0.d0
      x(3,1) = 0.d0
      v(1,1) = 0.d0
      v(2,1) = 0.d0
      v(3,1) = 0.d0
```

```
c
c Calculate coordinates and velocities of the central body
      do j = 2, nbod
        mtot = mtot  +  m(j)
        x(1,1) = x(1,1)  +  m(j) * xh(1,j)
        x(2,1) = x(2,1)  +  m(j) * xh(2,j)
        x(3,1) = x(3,1)  +  m(j) * xh(3,j)
        v(1,1) = v(1,1)  +  m(j) * vh(1,j)
        v(2,1) = v(2,1)  +  m(j) * vh(2,j)
        v(3,1) = v(3,1)  +  m(j) * vh(3,j)
      enddo
c
      temp = -1.d0 / (mtot + m(1))
      x(1,1) = temp * x(1,1)
      x(2,1) = temp * x(2,1)
      x(3,1) = temp * x(3,1)
      v(1,1) = temp * v(1,1)
      v(2,1) = temp * v(2,1)
      v(3,1) = temp * v(3,1)
c
c Calculate the barycentric coordinates and velocities
      do j = 2, nbod
        x(1,j) = xh(1,j) + x(1,1)
        x(2,j) = xh(2,j) + x(2,1)
        x(3,j) = xh(3,j) + x(3,1)
        v(1,j) = vh(1,j) + v(1,1)
```

```
            v(2,j) = vh(2,j) + v(2,1)
            v(3,j) = vh(3,j) + v(3,1)
          enddo
c
c
     ------------------------------------------------------------------
c
          return
          end
c
c
     %%%%
```

```
c
      subroutine mco_iden (time,jcen,nbod,nbig,h,m,xh,vh,x
     ,v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh
     (3,nbod)
      real*8 vh(3,nbod),ngf(4,nbod)
c
c Local
      integer j
c
c
c
      do j = 1, nbod
        x(1,j) = xh(1,j)
        x(2,j) = xh(2,j)
        x(3,j) = xh(3,j)
```

```
            v(1,j) = vh(1,j)
            v(2,j) = vh(2,j)
            v(3,j) = vh(3,j)
         enddo
c
c
c-----------------------------------------------------------------------
c
         return
         end
c
c
c%%%
```

```
c
c
      ------------------------------------------------------------------

c
      subroutine mco_x2a (gm,x,y,z,u,v,w,a,r,v2)
c
      implicit none
c
c Input/Output
      real*8 gm,x,y,z,u,v,w,a,r,v2
c
c
      ------------------------------------------------------------------

c
      r  = dsqrt(x * x  +  y * y  +  z * z)
      v2 =       u * u  +  v * v  +  w * w
      a  = gm * r / (2.d0 * gm  -  r * v2)
c
c
      ------------------------------------------------------------------

c
      return
      end
```

```
c
c
%%%
```

```
c
c-----------------------------------------------------------------------
c
      subroutine mco_x2ov (rcen,rmax,mcen,m,x,y,z,u,v,w,fr
     &   ,theta,phi,fv,
     %   vtheta,vphi)
c
      implicit none
      include 'mercury.inc'
c
c  Input/Output
      real*8 rcen,rmax,mcen,m,x,y,z,u,v,w,fr,theta,phi,fv,
     &   vtheta,vphi
c
c  Local
      real*8 r,v2,v1,be,ke,temp
c
c
c-----------------------------------------------------------------------
c
        r = dsqrt(x*x + y*y + z*z)
        v2 =      u*u + v*v + w*w
        v1 = dsqrt(v2)
        be = (mcen + m) / r
```

```
      ke = .5d0 * v2
c
      fr = log10 (min(max(r, rcen), rmax) / rcen)
      temp = ke / be
      fv = 1.d0 / (1.d0 + 2.d0*temp*temp)
c
      theta  = mod (acos (z / r) + TWOPI, TWOPI)
      vtheta = mod (acos (w / v1) + TWOPI, TWOPI)
      phi  = mod (atan2 (y, x) + TWOPI, TWOPI)
      vphi = mod (atan2 (v, u) + TWOPI, TWOPI)
c
c
c     ------------------------------------------------------------------
c
      return
      end
c
c
c%%%
```

```
c
c     This special version is for MIO_OUT.FOR to record
     spin into element output file
c Author: John E. Chambers / Yu-Cian Hong
c
c Calculates output variables for an object given its
     coordinates and
c velocities. The output variables are:
c  r = the radial distance
c  theta = polar angle

c  phi = azimuthal angle
c  fv = 1 / [1 + 2(ke/be)^2], where be and ke are the
     object's binding and
c                                kinetic energies. (Note that
      0 < fv < 1).
c  vtheta = polar angle of velocity vector
c  vphi = azimuthal angle of the velocity vector
c
c
      ------------------------------------------------------------------

c
        subroutine mco_x2ov_s (rcen,rmax,mcen,m,x,y,z,u,v,w,
     %  sx,sy,sz,fr,theta,phi,fv,vtheta,vphi,stheta,sphi)
```

```
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      real*8 rcen,rmax,mcen,m,x,y,z,u,v,w,fr,theta,phi,fv,
     &   vtheta,vphi
      real*8 sx,sy,sz,stheta,sphi
c
c Local
      real*8 r,v2,v1,be,ke,temp
c
c
     ------------------------------------------------------------------
c
        r = dsqrt(x*x + y*y + z*z)
        v2 =      u*u + v*v + w*w
        v1 = dsqrt(v2)
        be = (mcen + m) / r
        ke = .5d0 * v2
c
        fr = log10 (min(max(r, rcen), rmax) / rcen)
        temp = ke / be
        fv = 1.d0 / (1.d0 + 2.d0*temp*temp)
c
```

```
          theta  = mod (acos (z / r) + TWOPI, TWOPI)
          vtheta = mod (acos (w / v1) + TWOPI, TWOPI)
          stheta = mod (acos (sz) + TWOPI, TWOPI)
          phi  = mod (atan2 (y, x) + TWOPI, TWOPI)
          vphi = mod (atan2 (v, u) + TWOPI, TWOPI)
          sphi = mod (atan2 (sy, sx) + TWOPI, TWOPI)
c
c
     ------------------------------------------------------------------

c
          return
          end
c
c
     %%%%
```

```
c Integrates NBOD bodies (of which NBIG are Big) for one
   timestep H0
c using the Bulirsch-Stoer method. The accelerations are
   calculated using the
c subroutine FORCE. The accuracy of the step is
   approximately determined
c by the tolerance parameter TOL.
c
c N.B. Input/output must be in coordinates with respect to
    the central body.
c ===
c
c
c-----------------------------------------------------------------------
c
      subroutine mdt_bs1 (time,h0,hdid,tol,jcen,nbod,nbig,
     %   mass,x0,v0,s,
     %   rphys,rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce
     %   ,force,dsdt)
c
      implicit none
      include 'mercury.inc'
c
      real*8 SHRINK,GROW
      parameter (SHRINK=.55d0,GROW=1.3d0)
```

```
c
c Input/Output
      integer nbod, nbig, opt(8), stat(nbod), dtflag,
     &   ngflag
      integer nce, ice(nce), jce(nce)
      real*8 time,h0,hdid,tol,jcen(3),mass(nbod),x0(3,nbod
     &   ),v0(3,nbod)
      real*8 s(3,nbod),ngf(4,nbod),rphys(nbod),rcrit(nbod)
     &   ,dsdt(3,nbod)
      external force
c
c Local
      integer j, j1, k, n
      real*8 tmp0,tmp1,tmp2,errmax,tol2,h,hx2,h2(8)
      real*8 x(3,NMAX),v(3,NMAX),xend(3,NMAX),vend(3,NMAX)
     &   ,send(3,NMAX)
      real*8 a(3,NMAX),a0(3,NMAX),d(6,NMAX,8),xscal(NMAX),
     &   vscal(NMAX)
      real*8 s0(3,NMAX),ds0dt(3,NMAX),dummy(nbod),ltmp(3),
     &   kk(3)
c
c
      ------------------------------------------------------------------------
c
      tol2 = tol * tol
```

```
c
c Calculate arrays used to scale the relative error (R^2
   for position and
c V^2 for velocity).
      do k = 2, nbod
        tmp1 = x0(1,k)*x0(1,k) + x0(2,k)*x0(2,k) + x0(3,k)
           *x0(3,k)
        tmp2 = v0(1,k)*v0(1,k) + v0(2,k)*v0(2,k) + v0(3,k)
           *v0(3,k)
        xscal(k) = 1.d0 / tmp1
        vscal(k) = 1.d0 / tmp2
      end do
c
      call spin (nbod,nbig,mass,x0,v0,s,dsdt)
      ds0dt = dsdt
      s0 = s
c Calculate accelerations at the start of the step
      call force (time,jcen,nbod,nbig,mass,x0,v0,s0,rcrit,
         a0,stat,ngf,
     %  ngflag,opt,nce,ice,jce,dsdt)
c
 100  continue
c
c For each value of N, do a modified-midpoint integration
   with 2N substeps
      do n = 1, 8
```

```
      h = h0 / (2.d0 * float(n))
      h2(n) = .25d0 / (n*n)
      hx2 = h * 2.d0
c
      do k = 2, nbod
        x(1,k) = x0(1,k) + h*v0(1,k)
        x(2,k) = x0(2,k) + h*v0(2,k)
        x(3,k) = x0(3,k) + h*v0(3,k)
c         s(1,k) = s0(1,k) + h*ds0dt(1,k)
c         s(2,k) = s0(2,k) + h*ds0dt(2,k)
c         s(3,k) = s0(3,k) + h*ds0dt(3,k)
        v(1,k) = v0(1,k) + h*a0(1,k)
        v(2,k) = v0(2,k) + h*a0(2,k)
        v(3,k) = v0(3,k) + h*a0(3,k)
      end do
      call spin (nbod,nbig,mass,x,v,s,dsdt)
      call force (time,jcen,nbod,nbig,mass,x,v,s0,rcrit,
     a,stat,ngf,
     %  ngflag,opt,nce,ice,jce,dsdt)
      do k = 2, nbod
        xend(1,k) = x0(1,k) + hx2*v(1,k)
        xend(2,k) = x0(2,k) + hx2*v(2,k)
        xend(3,k) = x0(3,k) + hx2*v(3,k)
c         send(1,k) = s0(1,k) + hx2*dsdt(1,k)
c         send(2,k) = s0(2,k) + hx2*dsdt(2,k)
c         send(3,k) = s0(3,k) + hx2*dsdt(3,k)
```

```
            vend(1,k) = v0(1,k) + hx2*a(1,k)
            vend(2,k) = v0(2,k) + hx2*a(2,k)
            vend(3,k) = v0(3,k) + hx2*a(3,k)
          end do
c
          do j = 2, n
            call force (time,jcen,nbod,nbig,mass,xend,vend,
               s0,rcrit,a,
     %                  stat,ngf,ngflag,opt,nce,ice,jce,dsdt
          )
            do k = 2, nbod
              x(1,k) = x(1,k) + hx2*vend(1,k)
              x(2,k) = x(2,k) + hx2*vend(2,k)
              x(3,k) = x(3,k) + hx2*vend(3,k)
c             s(1,k) = s(1,k) + hx2*dsdt(1,k)
c             s(2,k) = s(2,k) + hx2*dsdt(2,k)
c             s(3,k) = s(3,k) + hx2*dsdt(3,k)
              v(1,k) = v(1,k) + hx2*a(1,k)
              v(2,k) = v(2,k) + hx2*a(2,k)
              v(3,k) = v(3,k) + hx2*a(3,k)
            end do
            call force (time,jcen,nbod,nbig,mass,x,v,s0,
               rcrit,a,stat,ngf,
     %         ngflag,opt,nce,ice,jce,dsdt)
            do k = 2, nbod
              xend(1,k) = xend(1,k) + hx2*v(1,k)
```

```
            xend(2,k) = xend(2,k) + hx2*v(2,k)
            xend(3,k) = xend(3,k) + hx2*v(3,k)
c             send(1,k) = send(1,k) + hx2*dsdt(1,k)
c             send(2,k) = send(2,k) + hx2*dsdt(2,k)
c             send(3,k) = send(3,k) + hx2*dsdt(3,k)
            vend(1,k) = vend(1,k) + hx2*a(1,k)
            vend(2,k) = vend(2,k) + hx2*a(2,k)
            vend(3,k) = vend(3,k) + hx2*a(3,k)
          end do
        end do
c
        call force (time,jcen,nbod,nbig,mass,xend,vend,s0,
           rcrit,a,
     %              stat,ngf,ngflag,opt,nce,ice,jce,dsdt)
c
        do k = 2, nbod
          d(1,k,n) = .5d0*(xend(1,k) + x(1,k) + h*vend(1,k
             ))
          d(2,k,n) = .5d0*(xend(2,k) + x(2,k) + h*vend(2,k
             ))
          d(3,k,n) = .5d0*(xend(3,k) + x(3,k) + h*vend(3,k
             ))
          d(4,k,n) = .5d0*(vend(1,k) + v(1,k) + h*a(1,k))
          d(5,k,n) = .5d0*(vend(2,k) + v(2,k) + h*a(2,k))
          d(6,k,n) = .5d0*(vend(3,k) + v(3,k) + h*a(3,k))
        end do
```

```
c
c Update the D array, used for polynomial extrapolation
        do j = n - 1, 1, -1
          j1 = j + 1
          tmp0 = 1.d0 / (h2(j) - h2(n))
          tmp1 = tmp0 * h2(j1)
          tmp2 = tmp0 * h2(n)
          do k = 2, nbod
            d(1,k,j) = tmp1 * d(1,k,j1) - tmp2 * d(1,k,j
               )
            d(2,k,j) = tmp1 * d(2,k,j1) - tmp2 * d(2,k,j
               )
            d(3,k,j) = tmp1 * d(3,k,j1) - tmp2 * d(3,k,j
               )
            d(4,k,j) = tmp1 * d(4,k,j1) - tmp2 * d(4,k,j
               )
            d(5,k,j) = tmp1 * d(5,k,j1) - tmp2 * d(5,k,j
               )
            d(6,k,j) = tmp1 * d(6,k,j1) - tmp2 * d(6,k,j
               )
          end do
        end do
c
c After several integrations, test the relative error on
   extrapolated values
        if (n.gt.3) then
```

```
            errmax = 0.d0
c
c Maximum relative position and velocity errors (last D
     term added)
            do k = 2, nbod
              tmp1 = max( d(1,k,1)*d(1,k,1), d(2,k,1)*d(2,k
                 ,1),
     %                    d(3,k,1)*d(3,k,1) )
              tmp2 = max( d(4,k,1)*d(4,k,1), d(5,k,1)*d(5,k
                 ,1),
     %                    d(6,k,1)*d(6,k,1) )
              errmax = max(errmax, tmp1*xscal(k), tmp2*vscal
                 (k))
            end do
c
c If error is smaller than TOL, update position and
     velocity arrays, and exit
            if (errmax.le.tol2) then
              do k = 2, nbod
                x0(1,k) = d(1,k,1)
                x0(2,k) = d(2,k,1)
                x0(3,k) = d(3,k,1)
                v0(1,k) = d(4,k,1)
                v0(2,k) = d(5,k,1)
                v0(3,k) = d(6,k,1)
              end do
```

```
c
                do j = 2, n
                  do k = 2, nbod
                    x0(1,k) = x0(1,k) + d(1,k,j)
                    x0(2,k) = x0(2,k) + d(2,k,j)
                    x0(3,k) = x0(3,k) + d(3,k,j)
                    v0(1,k) = v0(1,k) + d(4,k,j)
                    v0(2,k) = v0(2,k) + d(5,k,j)
                    v0(3,k) = v0(3,k) + d(6,k,j)
                  end do
                end do
c


c Update spin and normalize
               do k = 2, nbod
                s(1,k) = s0(1,k) + h0*ds0dt(1,k)
                s(2,k) = s0(2,k) + h0*ds0dt(2,k)
                s(3,k) = s0(3,k) + h0*ds0dt(3,k)
                if (k.eq.5) then
                end if
                tmp1 = s(1,k)**2. + s(2,k)**2. + s(3,k)**2.
                tmp2 = dsqrt(tmp1)
                if (k.eq.5) then
                end if
                s(1,k) = s(1,k) /tmp2
                s(2,k) = s(2,k) /tmp2
```

```
            s(3,k) = s(3,k) /tmp2
c            kk(1) = s(1,k) / tmp2
c            kk(2) = s(2,k) / tmp2
c            kk(3) = s(3,k) / tmp2
            if (k.eq.5) then
c              tmp1 = 1.90766255900899276d-004 /
   2.95912357613654247d-004
            end if
          end do
c check obliquity evolution
            k = 2
            ltmp(1)  = x0(2,k) * v0(3,k) - x0(3,k) * v0(2,
               k)
            ltmp(2)  = x0(3,k) * v0(1,k) - x0(1,k) * v0(3,
               k)
            ltmp(3)  = x0(1,k) * v0(2,k) - x0(2,k) * v0(1,
               k)
            tmp1 = ltmp(1)**2. + ltmp(2)**2. + ltmp(3)**2.
            tmp2 = dsqrt(tmp1)
            ltmp(1) = ltmp(1)/tmp2
            ltmp(2) = ltmp(2)/tmp2
            ltmp(3) = ltmp(3)/tmp2
            tmp0 = ltmp(1)*s(1,k) + ltmp(2)*s(2,k) + ltmp
               (3)*s(3,k)
            tmp1 = mod(time, 36525.)
            tmp1 = abs(tmp1)
```

```
c Save the actual stepsize used
            hdid = h0
c
c Recommend a new stepsize for the next call to this
   subroutine
            if (n.eq.8) h0 = h0 * SHRINK
            if (n.lt.7) h0 = h0 * GROW
            return
          end if
        end if
c
      end do
c
c If errors were too large, redo the step with half the
   previous step size.
      h0 = h0 * .5d0
      goto 100
c
c
   ------------------------------------------------------------------------
c
      end
c
c
   %%%%
```

```
c
c       MDT_BS2.FOR    (May 2019)
c
c
%%%
```

```
c ===
c
c
   ------------------------------------------------------------------------


c
      subroutine mdt_bs2 (time,h0,hdid,tol,jcen,nbod,nbig,
     %   mass,x0,v0,s,
     %  rphys,rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce
     %  ,force,dsdt)
c
      implicit none
      include 'mercury.inc'
c
      real*8 SHRINK,GROW
      parameter (SHRINK=.55d0,GROW=1.3d0)
c
c Input/Output
      integer nbod, nbig, opt(8), stat(nbod), dtflag,
     %   ngflag
      real*8 time,h0,hdid,tol,jcen(3),mass(nbod),x0(3,nbod
     %   ),v0(3,nbod)
      real*8 s(3,nbod),ngf(4,nbod),rphys(nbod),rcrit(nbod)
     %   ,dsdt(3,nbod)
      integer nce,ice(nce),jce(nce)
      external force
```

```
c
c Local
      integer j,j1,k,nn
      real*8 tmp0,tmp1,tmp2,errmax,tol2,h,h2(12),hby2,
     h2by2
      real*8 xend(3,NMAX),b(3,NMAX),c(3,NMAX),s0(3,NMAX),
     ds0dt(3,NMAX)
      real*8 a(3,NMAX),a0(3,NMAX),d(6,NMAX,12),xscal(NMAX)
     ,vscal(NMAX)
c
c
```

```
c
c     to guard the value of hdid
      tol2 = tol * tol
c
c Calculate arrays used to scale the relative error (R^2
   for position and
c V^2 for velocity).
      do k = 2, nbod
        tmp1 = x0(1,k)*x0(1,k) + x0(2,k)*x0(2,k) + x0(3,k)
     *x0(3,k)
        tmp2 = v0(1,k)*v0(1,k) + v0(2,k)*v0(2,k) + v0(3,k)
     *v0(3,k)
        xscal(k) = 1.d0 / tmp1
```

```
            vscal(k) = 1.d0 / tmp2
          end do
c Calculate accelerations at the start of the step
          call spin (nbod,nbig,mass,x0,v0,s,dsdt)
          ds0dt = dsdt*1.d0
          s0 = s*1.d0
          call force (time,jcen,nbod,nbig,mass,x0,v0,s0,rcrit,
             a0,stat,ngf,
     %  ngflag,opt,nce,ice,jce,dsdt)
c
 100   continue
c
c For each value of N, do a modified-midpoint integration
   with N substeps
          do nn = 1, 12
c           restore epoch for each n
            h = h0 / (dble(nn))
            hby2  = .5d0 * h
            h2(nn) = h * h
            h2by2 = .5d0 * h2(nn)
c
            do k = 2, nbod
              b(1,k) = .5d0*a0(1,k)
              b(2,k) = .5d0*a0(2,k)
              b(3,k) = .5d0*a0(3,k)
              c(1,k) = 0.d0
```

```
            c(2,k) = 0.d0
            c(3,k) = 0.d0
            xend(1,k) = h2by2 * a0(1,k)  +  h * v0(1,k)  +
               x0(1,k)
            xend(2,k) = h2by2 * a0(2,k)  +  h * v0(2,k)  +
               x0(2,k)
            xend(3,k) = h2by2 * a0(3,k)  +  h * v0(3,k)  +
               x0(3,k)
          end do
c
          do j = 2, nn
            tmp0 = h * dble(j)
            call force (time,jcen,nbod,nbig,mass,xend,v0,s0,
               rcrit,a,stat,
     %        ngf,ngflag,opt,nce,ice,jce,dsdt)
            do k = 2, nbod
              b(1,k) = b(1,k) + a(1,k)
              b(2,k) = b(2,k) + a(2,k)
              b(3,k) = b(3,k) + a(3,k)
              c(1,k) = c(1,k) + b(1,k)
              c(2,k) = c(2,k) + b(2,k)
              c(3,k) = c(3,k) + b(3,k)
              xend(1,k) = h2(nn)*c(1,k) + h2by2*a0(1,k) +
                 tmp0*v0(1,k)
     %                  + x0(1,k)
              xend(2,k) = h2(nn)*c(2,k) + h2by2*a0(2,k) +
```

```
                tmp0*v0(2,k)
     %                  + x0(2,k)
              xend(3,k) = h2(nn)*c(3,k) + h2by2*a0(3,k) +
                tmp0*v0(3,k)
     %                  + x0(3,k)
            end do
          end do
c
c         advance velocity in 1 step
            call force (time,jcen,nbod,nbig,mass,xend,v0,s0,
              rcrit,a,stat,
     %        ngf,ngflag,opt,nce,ice,jce,dsdt)
c
          do k = 2, nbod
            d(1,k,nn) = xend(1,k)
            d(2,k,nn) = xend(2,k)
            d(3,k,nn) = xend(3,k)
            d(4,k,nn) = h*b(1,k) + hby2*a(1,k) + v0(1,k)
            d(5,k,nn) = h*b(2,k) + hby2*a(2,k) + v0(2,k)
            d(6,k,nn) = h*b(3,k) + hby2*a(3,k) + v0(3,k)
          end do
c
c Update the D array, used for polynomial extrapolation
          do j = nn - 1, 1, -1
            j1 = j + 1
            tmp0 = 1.d0 / (h2(j) - h2(nn))
```

```
            tmp1 = tmp0 * h2(j1)
            tmp2 = tmp0 * h2(nn)
            do k = 2, nbod
              d(1,k,j) = tmp1 * d(1,k,j1) - tmp2 * d(1,k,j
                 )
              d(2,k,j) = tmp1 * d(2,k,j1) - tmp2 * d(2,k,j
                 )
              d(3,k,j) = tmp1 * d(3,k,j1) - tmp2 * d(3,k,j
                 )
              d(4,k,j) = tmp1 * d(4,k,j1) - tmp2 * d(4,k,j
                 )
              d(5,k,j) = tmp1 * d(5,k,j1) - tmp2 * d(5,k,j
                 )
              d(6,k,j) = tmp1 * d(6,k,j1) - tmp2 * d(6,k,j
                 )
            end do
          end do
c
c After several integrations, test the relative error on
   extrapolated values
          if (nn.gt.3) then
            errmax = 0.d0
c
c Maximum relative position and velocity errors (last D
   term added)
            do k = 2, nbod
```

```
            tmp1 = max( d(1,k,1)*d(1,k,1), d(2,k,1)*d(2,k
               ,1),
     %                  d(3,k,1)*d(3,k,1) )
            tmp2 = max( d(4,k,1)*d(4,k,1), d(5,k,1)*d(5,k
               ,1),
     %                  d(6,k,1)*d(6,k,1) )
            errmax = max( errmax, tmp1*xscal(k), tmp2*
               vscal(k) )
          end do
c
c If error is smaller than TOL, update position and
   velocity arrays and exit
          if (errmax.le.tol2) then
            do k = 2, nbod
              x0(1,k) = d(1,k,1)
              x0(2,k) = d(2,k,1)
              x0(3,k) = d(3,k,1)
              v0(1,k) = d(4,k,1)
              v0(2,k) = d(5,k,1)
              v0(3,k) = d(6,k,1)
            end do
c
            do j = 2, nn
              do k = 2, nbod
                x0(1,k) = x0(1,k) + d(1,k,j)
                x0(2,k) = x0(2,k) + d(2,k,j)
```

```
                x0(3,k) = x0(3,k) + d(3,k,j)
                v0(1,k) = v0(1,k) + d(4,k,j)
                v0(2,k) = v0(2,k) + d(5,k,j)
                v0(3,k) = v0(3,k) + d(6,k,j)
              end do
            end do
c
c Update spin and normalize
            do k = 2, nbod
             s(1,k) = (s0(1,k) + h0*ds0dt(1,k))*1.d0
             s(2,k) = (s0(2,k) + h0*ds0dt(2,k))*1.d0
             s(3,k) = (s0(3,k) + h0*ds0dt(3,k))*1.d0
             tmp1 = (s(1,k)**2. + s(2,k)**2. + s(3,k)**2.)
                *1.d0
             tmp2 = (dsqrt(tmp1))*1.d0
             s(1,k) = s(1,k) /tmp2*1.d0
             s(2,k) = s(2,k) /tmp2*1.d0
             s(3,k) = s(3,k) /tmp2*1.d0
            end do
c Save the actual stepsize used
             hdid = h0
c
c Recommend a new stepsize for the next call to this
   subroutine
             if (nn.ge.8) h0 = h0 * SHRINK
             if (nn.lt.7) h0 = h0 * GROW
```

```
            return
          end if
        end if
c
      end do
c
c If errors were too large, redo the step with half the
   previous step size.
      h0 = h0 * .5d0
      goto 100
c
c
   ------------------------------------------------------------------------
c
      end
c
c
   %%%%
```

```
c
c Author: John E. Chambers / Yu-Cian Hong
c
c Calculates accelerations on a set of NBOD bodies (of
   which NBIG are Big)
c due to Newtonian gravitational perturbations, post-
   Newtonian
c corrections (if required), cometary non-gravitational
   forces (if required)
c and user-defined forces (if required).
c
c N.B. Input/output must be in coordinates with respect to
    the central body.
c ===
c
c
   ------------------------------------------------------------------

c
      subroutine mfo_all (time,jcen,nbod,nbig,m,x,v,s,
         rcrit,a,stat,ngf,
     %  ngflag,opt,nce,ice,jce,dsdt)
c
      implicit none
      include 'mercury.inc'
c
```

```
c Input/Output
      integer nbod,nbig,ngflag,stat(nbod),opt(8),nce,ice(
     nce),jce(nce)
      real*8 time,jcen(3),m(nbod),x(3,nbod),v(3,nbod),s(3,
     nbod)
      real*8 a(3,nbod),ngf(4,nbod),rcrit(nbod),dsdt(3,nbod
     )
c
c Local
      integer j
      real*8 acor(3,NMAX),acen(3),dummy
c
c
------------------------------------------------------------------------

c
c Newtonian gravitational forces
      call mfo_grav (nbod,nbig,m,x,v,a,stat)
c
c Correct for oblateness of the central body
c       if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0)
    then
c         call mfo_obl (jcen,nbod,m,x,acor,acen)
c         do j = 2, nbod
c           a(1,j) = a(1,j) + (acor(1,j) - acen(1))
c           a(2,j) = a(2,j) + (acor(2,j) - acen(2))
```

```
c           a(3,j) = a(3,j) + (acor(3,j) - acen(3))
c         end do
c       end if
c
c Include non-gravitational (cometary jet) accelerations
   if necessary
c       if (ngflag.eq.1.or.ngflag.eq.3) then
c         call mfo_ngf (nbod,x,v,acor,ngf)
c         do j = 2, nbod
c           a(1,j) = a(1,j) + acor(1,j)
c           a(2,j) = a(2,j) + acor(2,j)
c           a(3,j) = a(3,j) + acor(3,j)
c         end do
c       end if
c
c Include radiation pressure/Poynting-Robertson drag if
   necessary
c       if (ngflag.eq.2.or.ngflag.eq.3) then
c         call mfo_pr (nbod,nbig,m,x,v,acor,ngf)
c         do j = 2, nbod
c           a(1,j) = a(1,j) + acor(1,j)
c           a(2,j) = a(2,j) + acor(2,j)
c           a(3,j) = a(3,j) + acor(3,j)
c         end do
c       end if
c
```

```
c Include post-Newtonian corrections if required
c       if (opt(7).eq.1) then
c         call mfo_pn (nbod,nbig,m,x,v,acor)
c         do j = 2, nbod
c           a(1,j) = a(1,j) + acor(1,j)
c           a(2,j) = a(2,j) + acor(2,j)
c           a(3,j) = a(3,j) + acor(3,j)
c        end do
c       end if
c
c Include user-defined accelerations if required
      if (opt(8).eq.1) then
        call mfo_user (time,jcen,nbod,nbig,m,x,v,acor,s)
        do j = 2, nbod
          a(1,j) = a(1,j) + acor(1,j)
          a(2,j) = a(2,j) + acor(2,j)
          a(3,j) = a(3,j) + acor(3,j)
        end do
      end if
c
c
      ------------------------------------------------------------------------


c
      return
      end
```

```
c
c
c%%%
```

```
     to central body!!!!
c ===
c
     ------------------------------------------------------------------


c
      subroutine mfo_grav (nbod,nbig,m,x,v,a,stat)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod)
      real*8 m(nbod), x(3,nbod), v(3,nbod), a(3,nbod)
c
c Local
      integer i, j
      real*8 sx, sy, sz, dx, dy, dz, tmp1, tmp2, s_1, s2,
     &   s_3, r3(NMAX)
c
c
     ------------------------------------------------------------------


c
      sx = 0.d0
      sy = 0.d0
```

```
      sz = 0.d0
      do i = 2, nbod
        a(1,i) = 0.d0
        a(2,i) = 0.d0
        a(3,i) = 0.d0
        s2 = x(1,i)*x(1,i) + x(2,i)*x(2,i) + x(3,i)*x(3,i)
        s_1  = 1.d0 / dsqrt(s2)
        r3(i) = s_1 * s_1 * s_1
      end do
c
      do i = 2, nbod
        tmp1 = m(i) * r3(i)
        sx = sx  -  tmp1 * x(1,i)
        sy = sy  -  tmp1 * x(2,i)
        sz = sz  -  tmp1 * x(3,i)
      end do
c
c Direct terms
      do i = 2, nbig
        do j = i + 1, nbod
          dx = x(1,j) - x(1,i)
          dy = x(2,j) - x(2,i)
          dz = x(3,j) - x(3,i)
          s2 = dx*dx + dy*dy + dz*dz
          s_1 = 1.d0 / dsqrt(s2)
          s_3 = s_1 * s_1 * s_1
```

```
            tmp1 = s_3 * m(i)
            tmp2 = s_3 * m(j)
            a(1,j) = a(1,j) - tmp1 * dx
            a(2,j) = a(2,j) - tmp1 * dy
            a(3,j) = a(3,j) - tmp1 * dz
            a(1,i) = a(1,i) + tmp2 * dx
            a(2,i) = a(2,i) + tmp2 * dy
            a(3,i) = a(3,i) + tmp2 * dz
          end do
        end do
c
c Indirect terms (add these on last to reduce roundoff
   error)
        do i = 2, nbod
          tmp1 = m(1) * r3(i)
          a(1,i) = a(1,i) + sx - tmp1 * x(1,i)
          a(2,i) = a(2,i) + sy - tmp1 * x(2,i)
          a(3,i) = a(3,i) + sz - tmp1 * x(3,i)
        end do
c
c
   ------------------------------------------------------------------------


c
        return
        end
```

```
c
c
c%%%
```

```
     the output
c variables of the objects at this time. The output
     variables are:
c expressed as
c    r = the radial distance
c    theta = polar angle
c    phi = azimuthal angle
c    fv = 1 / [1 + 2(ke/be)^2], where be and ke are the
     object's binding and
c                                kinetic energies. (Note that
      0 < fv < 1).
c    vtheta = polar angle of velocity vector
c    vphi = azimuthal angle of the velocity vector
c
c
     ------------------------------------------------------------------------


c
      subroutine mio_ce (time,tstart,rcen,rmax,nbod,nbig,m
     ,stat,id,
     %  nclo,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,
     jxvclo,mem,
     %  lmem,outfile,nstored,ceflush)
c
      implicit none
      include 'mercury.inc'
```

```
c
c Input/Output
      integer nbod,nbig,opt(8),stat(nbod),lmem(NMESS),
     &   stopflag
      integer nclo,iclo(nclo),jclo(nclo),nstored,ceflush
      real*8 time,tstart,rcen,rmax,m(nbod),tclo(nclo),dclo
     &   (nclo)
      real*8 ixvclo(6,nclo),jxvclo(6,nclo)
      character*80 outfile(3),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer k,year,month
      real*8 tmp0,t1,rfac,fr,fv,theta,phi,vtheta,vphi
      character*80 c(200)
      character*38 fstop
      character*8 mio_fl2c, mio_re2c
      character*6 tstring
c
c
      ------------------------------------------------------------------------
c
      save c
c
c Scaling factor (maximum possible range) for distances
```

```
      rfac = log10 (rmax / rcen)
c
c Store details of each new close-encounter minimum
      do k = 1, nclo
        nstored = nstored + 1
        c(nstored)(1:8)   = mio_fl2c(tclo(k))
        c(nstored)(9:16)  = mio_re2c(dble(iclo(k)-1),0.d0
           ,11239423.99d0)
        c(nstored)(12:19) = mio_re2c(dble(jclo(k)-1),0.d0
           ,11239423.99d0)
        c(nstored)(15:22) = mio_fl2c(dclo(k))
c
        call mco_x2ov (rcen,rmax,m(1),0.d0,ixvclo(1,k),
           ixvclo(2,k),
     %     ixvclo(3,k),ixvclo(4,k),ixvclo(5,k),ixvclo(6,k),
           fr,theta,phi,
     %     fv,vtheta,vphi)
        c(nstored)(23:30) = mio_re2c (fr     , 0.d0, rfac)
        c(nstored)(27:34) = mio_re2c (theta  , 0.d0, PI)
        c(nstored)(31:38) = mio_re2c (phi    , 0.d0, TWOPI)
        c(nstored)(35:42) = mio_re2c (fv     , 0.d0, 1.d0)
        c(nstored)(39:46) = mio_re2c (vtheta , 0.d0, PI)
        c(nstored)(43:50) = mio_re2c (vphi   , 0.d0, TWOPI)
c
        call mco_x2ov (rcen,rmax,m(1),0.d0,jxvclo(1,k),
           jxvclo(2,k),
```

```
%     jxvclo(3,k),jxvclo(4,k),jxvclo(5,k),jxvclo(6,k),
     fr,theta,phi,
%     fv,vtheta,vphi)
      c(nstored)(47:54) = mio_re2c (fr     , 0.d0, rfac)
      c(nstored)(51:58) = mio_re2c (theta  , 0.d0, PI)
      c(nstored)(55:62) = mio_re2c (phi    , 0.d0, TWOPI)
      c(nstored)(59:66) = mio_re2c (fv     , 0.d0, 1.d0)
      c(nstored)(63:74) = mio_re2c (vtheta, 0.d0, PI)
      c(nstored)(67:78) = mio_re2c (vphi   , 0.d0, TWOPI)
    end do
c
c If required, output the stored close encounter details
     if (nstored.ge.100.or.ceflush.eq.0) then
 10    open (22, file=outfile(2), status='old', access='
    append',err=10)
      do k = 1, nstored
        write (22,'(a1,a2,a70)') char(12),'6b',c(k)
           (1:70)
      end do
      close (22)
      nstored = 0
    end if
c
c If new encounter minima have occurred, decide whether to
   stop integration
     stopflag = 0
```

```
      if (opt(1).eq.1.and.nclo.gt.0) then
 20     open (23, file=outfile(3), status='old', access='
     append',err=20)
c If time style is Gregorian date then...
        tmp0 = tclo(1)
        if (opt(3).eq.1) then
          fstop = '(5a,/,9x,a,i10,1x,i2,1x,f4.1)'
          call mio_jd2y (tmp0,year,month,t1)
          write (23,fstop) mem(121)(1:lmem(121)),mem(126)
     %      (1:lmem(126)),id(iclo(1)),',',id(jclo(1)),
     %      mem(71)(1:lmem(71)),year,month,t1
c Otherwise...
        else
          if (opt(3).eq.3) then
            tstring = mem(2)
            fstop = '(5a,/,9x,a,f14.3,a)'
            t1 = (tmp0 - tstart) / 365.25d0
          else
            tstring = mem(1)
            fstop = '(5a,/,9x,a,f14.1,a)'
            if (opt(3).eq.0) t1 = tmp0
            if (opt(3).eq.2) t1 = tmp0 - tstart
          end if
          write (23,fstop) mem(121)(1:lmem(121)),mem(126)
     %      (1:lmem(126)),id(iclo(1)),',',id(jclo(1)),
     %      mem(71)(1:lmem(71)),t1,tstring
```

```
          end if
          stopflag = 1
          close(23)
        end if
c
c
     ------------------------------------------------------------------------

c
        return
        end
c
c
     %%%%
```

```
c parameters, to dump files. Also updates a restart file
   containing other
c variables used internally by MERCURY.
c
c
     ------------------------------------------------------------------

c
      subroutine mio_dump (time,tstart,tstop,dtout,algor,
         h0,tol,jcen,
     %   rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,x,v,s
         ,rho,rceh,
     %   stat,id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(nbod),opt(8),opflag,
         ndump,nfun
      integer lmem(NMESS)
      real*8 time,tstart,tstop,dtout,h0,tol,rmax,en(3),am
         (3)
      real*8 jcen(3),rcen,cefac,m(nbod),x(3,nbod),v(3,nbod
         )
      real*8 s(3,nbod),rho(nbod),rceh(nbod),ngf(4,nbod),
```

```
      epoch(nbod)
      character*80 dumpfile(4),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer idp,i,j,k,len,j1,j2
      real*8 rhocgs,k_2,rcen_2,rcen_4,rcen_6,x0(3,NMAX),v0
     (3,NMAX)
      character*150 c
c
c
     ------------------------------------------------------------------


c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
      rcen_4 = rcen_2 * rcen_2
      rcen_6 = rcen_4 * rcen_2
c
c If using close-binary star, convert to user coordinates
c       if (algor.eq.11) call mco_h2ub (time,jcen,nbod,nbig
     ,h0,m,x,v,
c      %    x0,v0)
c
c Dump to temporary files (idp=1) and real dump files (idp
```

```
   =2)
      do idp = 1, 2
c
c Dump data for the Big (i=1) and Small (i=2) bodies
        do i = 1, 2
          if (idp.eq.1) then
            if (i.eq.1) c(1:12) = 'big.tmp     '
            if (i.eq.2) c(1:12) = 'small.tmp   '
  20        open (31, file=c(1:12), status='unknown', err
     =20)
          else
  25        open (31, file=dumpfile(i), status='old', err
     =25)
          end if
c
c Write header lines, data style (and epoch for Big bodies
   )
          write (31,'(a)') mem(151+i)(1:lmem(151+i))
          if (i.eq.1) then
            j1 = 2
            j2 = nbig
          else
            j1 = nbig + 1
            j2 = nbod
          end if
          write (31,'(a)') mem(154)(1:lmem(154))
```

```
            write (31,'(a)') mem(155)(1:lmem(155))
            write (31,*) mem(156)(1:lmem(156)),'Cartesian'
            if (i.eq.1) write (31,*) mem(157)(1:lmem(157)),
               time
            write (31,'(a)') mem(155)(1:lmem(155))
c
c For each body...
            do j = j1, j2
              len = 37
              c(1:8) = id(j)
              write (c(9:37),'(1p,a3,e11.5,a3,e11.5)') ' r
                 =',rceh(j),
     %          ' d=',rho(j)/rhocgs
              if (m(j).gt.0) then
                write (c(len+1:len+25),'(a3,e22.15)') ' m=',
                   m(j)*k_2
                len = len + 25
              end if
              do k = 1, 3
                if (ngf(k,j).ne.0) then
                  write (c(len+1:len+16),'(a2,i1,a1,e12.5)')
                      ' a',k,'=',
     %              ngf(k,j)
                  len = len + 16
                end if
              end do
```

```
              if (ngf(4,j).ne.0) then
                write (c(len+1:len+15),'(a3,e12.5)') ' b=',
                   ngf(4,j)
                len = len + 15
              end if
              write (31,'(a)') c(1:len)
              if (algor.eq.11) then
                write (31,312) x0(1,j), x0(2,j), x0(3,j)
                write (31,312) v0(1,j), v0(2,j), v0(3,j)
              else
                write (31,312) x(1,j), x(2,j), x(3,j)
                write (31,312) v(1,j), v(2,j), v(3,j)
              end if
c              write (31,312) s(1,j)*k_2, s(2,j)*k_2, s(3,j)
   *k_2
              write (31,312) s(1,j), s(2,j), s(3,j)
            enddo
            close (31)
          end do
c
c Dump the integration parameters
  40      if (idp.eq.1) open (33,file='param.tmp',status='
     unknown',err=40)
  45      if (idp.eq.2) open (33, file=dumpfile(3), status='
     old', err=45)
c
```

```
c Important parameters
        write (33,'(a)') mem(151)(1:lmem(151))
        write (33,'(a)') mem(154)(1:lmem(154))
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,'(a)') mem(158)(1:lmem(158))
        write (33,'(a)') mem(155)(1:lmem(155))
        if (algor.eq.1) then
          write (33,*) mem(159)(1:lmem(159)),'MVS'
        else if (algor.eq.2) then
          write (33,*) mem(159)(1:lmem(159)),'BS'
        else if (algor.eq.3) then
          write (33,*) mem(159)(1:lmem(159)),'BS2'
        else if (algor.eq.4) then
          write (33,*) mem(159)(1:lmem(159)),'RADAU'
        else if (algor.eq.10) then
          write (33,*) mem(159)(1:lmem(159)),'HYBRID'
        else if (algor.eq.11) then
          write (33,*) mem(159)(1:lmem(159)),'CLOSE'
        else if (algor.eq.12) then
          write (33,*) mem(159)(1:lmem(159)),'WIDE'
        else
          write (33,*) mem(159)(1:lmem(159)),'0'
        end if
        write (33,*) mem(160)(1:lmem(160)),tstart
        write (33,*) mem(161)(1:lmem(161)),tstop
        write (33,*) mem(162)(1:lmem(162)),dtout
```

```
      write (33,*) mem(163)(1:lmem(163)),h0
      write (33,*) mem(164)(1:lmem(164)),tol
c
c Integration options
      write (33,'(a)') mem(155)(1:lmem(155))
      write (33,'(a)') mem(165)(1:lmem(165))
      write (33,'(a)') mem(155)(1:lmem(155))
      if (opt(1).eq.0) then
        write (33,'(2a)') mem(166)(1:lmem(166)),mem(5)
           (1:lmem(5))
      else
        write (33,'(2a)') mem(166)(1:lmem(166)),mem(6)
           (1:lmem(6))
      end if
      if (opt(2).eq.0) then
        write (33,'(2a)') mem(167)(1:lmem(167)),mem(5)
           (1:lmem(5))
        write (33,'(2a)') mem(168)(1:lmem(168)),mem(5)
           (1:lmem(5))
      else if (opt(2).eq.2) then
        write (33,'(2a)') mem(167)(1:lmem(167)),mem(6)
           (1:lmem(6))
        write (33,'(2a)') mem(168)(1:lmem(168)),mem(6)
           (1:lmem(6))
      else
        write (33,'(2a)') mem(167)(1:lmem(167)),mem(6)
```

```
      (1:lmem(6))
   write (33,'(2a)') mem(168)(1:lmem(168)),mem(5)
      (1:lmem(5))
end if
if (opt(3).eq.0.or.opt(3).eq.2) then
   write (33,'(2a)') mem(169)(1:lmem(169)),mem(1)
      (1:lmem(1))
else
   write (33,'(2a)') mem(169)(1:lmem(169)),mem(2)
      (1:lmem(2))
end if
if (opt(3).eq.2.or.opt(3).eq.3) then
   write (33,'(2a)') mem(170)(1:lmem(170)),mem(6)
      (1:lmem(6))
else
   write (33,'(2a)') mem(170)(1:lmem(170)),mem(5)
      (1:lmem(5))
end if
if (opt(4).eq.1) then
   write (33,'(2a)') mem(171)(1:lmem(171)),mem(7)
      (1:lmem(7))
else if (opt(4).eq.3) then
   write (33,'(2a)') mem(171)(1:lmem(171)),mem(9)
      (1:lmem(9))
else
   write (33,'(2a)') mem(171)(1:lmem(171)),mem(8)
```

```
          (1:lmem(8))
        end if
        write (33,'(a)') mem(172)(1:lmem(172))
        if (opt(7).eq.1) then
          write (33,'(2a)') mem(173)(1:lmem(173)),mem(6)
          (1:lmem(6))
        else
          write (33,'(2a)') mem(173)(1:lmem(173)),mem(5)
          (1:lmem(5))
        end if
        if (opt(8).eq.1) then
          write (33,'(2a)') mem(174)(1:lmem(174)),mem(6)
          (1:lmem(6))
        else
          write (33,'(2a)') mem(174)(1:lmem(174)),mem(5)
          (1:lmem(5))
        end if
c
c Infrequently-changed parameters
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,'(a)') mem(175)(1:lmem(175))
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,*) mem(176)(1:lmem(176)),rmax
        write (33,*) mem(177)(1:lmem(177)),rcen
        write (33,*) mem(178)(1:lmem(178)),m(1) * k_2
        write (33,*) mem(179)(1:lmem(179)),jcen(1) *
```

```
          rcen_2
          write (33,*) mem(180)(1:lmem(180)),jcen(2) *
          rcen_4
          write (33,*) mem(181)(1:lmem(181)),jcen(3) *
          rcen_6
          write (33,*) mem(182)(1:lmem(182))
          write (33,*) mem(183)(1:lmem(183))
          write (33,*) mem(184)(1:lmem(184)),cefac
          write (33,*) mem(185)(1:lmem(185)),ndump
          write (33,*) mem(186)(1:lmem(186)),nfun
          close (33)
c
c Create new version of the restart file
  60      if (idp.eq.1) open (35, file='restart.tmp', status
     ='unknown',
     %     err=60)
  65      if (idp.eq.2) open (35, file=dumpfile(4), status='
     old', err=65)
          write (35,'(1x,i2)') opflag
          write (35,*) en(1) * k_2
          write (35,*) am(1) * k_2
          write (35,*) en(3) * k_2
          write (35,*) am(3) * k_2
c          write (35,*) s(1,1) * k_2
c          write (35,*) s(2,1) * k_2
c         write (35,*) s(3,1) * k_2
```

```
          write (35,*) s(1,1)
          write (35,*) s(2,1)
          write (35,*) s(3,1)
          close (35)
        end do
c
c
     ------------------------------------------------------------------
c
 311   format (1x,a8,1x,a1,1p,e22.15,2(1x,e11.5))
 312   format (1p,3(1x,e22.15),1x,i8)
 313   format (1p,1x,e22.15,0p,2x,a)
 314   format (1x,a8,1x,a1,1p,e22.15,4(1x,e12.5),1x,e22
     .15,2(1x,e11.5))
        return
        end
c
c
     %%%%
```

```
c
c Author: John E. Chambers / Yu-Cian Hong
c
c Writes out an error message and terminates Mercury.
c
c
c-----------------------------------------------------------------------
c
      subroutine mio_err (unit,s1,ls1,s2,ls2,s3,ls3,s4,ls4
     :   )
c
      implicit none
c
c Input/Output
      integer unit,ls1,ls2,ls3,ls4
      character*80 s1,s2,s3,s4
c
c
c-----------------------------------------------------------------------
c
      write (*,'(/,2a)') ' ERROR: Programme terminated.
     :   See information'
     %  ,' file for details.'
```

```
c
      write (unit,'(/,3a,/,2a)') s1(1:ls1),s2(1:ls2),s3(1:
     &    ls3),
     %  ' ',s4(1:ls4)
      stop
c
c
c-----------------------------------------------------------------------
c
      end
c
c
c%%%
```

```
c string, using the new format compression:
c
c X is first converted to base 224, and then each base 224
    digit is converted
c to an ASCII character, such that 0 -> character 32, 1 ->
    character 33...
c and 223 -> character 255.
c The first 7 characters in the string are used to store
   the mantissa, and the
c eighth character is used for the exponent.
c
c ASCII characters 0 - 31 (CTRL characters) are not used,
   because they
c cause problems when using some operating systems.
c
c N.B. X must lie in the range -1.e112 < X < 1.e112
c ===
c
c
```

```
c
      function mio_fl2c (x)
c
      implicit none
c
```

```
c Input/Output
      real*8 x
      character*8 mio_fl2c
c
c Local
      integer ex
      real*8 ax,y
      character*8 mio_re2c
c
c
```

```
c
      if (x.eq.0) then
        y = .5d0
      else
        ax = abs(x)
        ex = int(log10(ax))
        if (ax.ge.1) ex = ex + 1
        y = ax*(10.d0**(-ex))
        if (y.eq.1) then
          y = y * .1d0
          ex = ex + 1
        end if
        y = sign(y,x) *.5d0 + .5d0
      end if
```

```
c
      mio_fl2c(1:8) = mio_re2c (y, 0.d0, 1.d0)
      ex = ex + 112
      if (ex.gt.223) ex = 223
      if (ex.lt.0) ex = 0
      mio_fl2c(8:8) = char(ex+32)
c
c
     ------------------------------------------------------------------------

c
      return
      end
c
c
%%%
```

```
c Reads names, masses, coordinates and velocities of all
   the bodies,
c and integration parameters for the MERCURY integrator
   package.
c If DUMPFILE(4) exists, the routine assumes this is a
   continuation of
c an old integration, and reads all the data from the dump
    files instead
c of the input files.
c
c N.B. All coordinates are with respect to the central
   body!!
c ===
c
c
   ------------------------------------------------------------------------


c
      subroutine mio_in (time,tstart,tstop,dtout,algor,h0,
         tol,rmax,rcen,
     %   jcen,en,am,cefac,ndump,nfun,nbod,nbig,m,x,v,s,rho,
         rceh,stat,id,
     %   epoch,ngf,opt,opflag,ngflag,outfile,dumpfile,lmem,
         mem)
c
      implicit none
```

```
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(NMAX),opt(8),opflag,
     &   ngflag
      integer lmem(NMESS),ndump,nfun
      real*8 time,tstart,tstop,dtout,h0,tol,rmax,rcen,jcen
     &   (3)
      real*8 en(3),am(3),m(NMAX),x(3,NMAX),v(3,NMAX),s(3,
     &   NMAX)
      real*8 rho(NMAX),rceh(NMAX),epoch(NMAX),ngf(4,NMAX),
     &   cefac
      character*80 outfile(3),dumpfile(4), mem(NMESS)
      character*8 id(NMAX)
c
c Local
      integer j,k,itmp,jtmp,informat,lim(2,10),nsub,year,
     &   month,lineno
      real*8 q,a,e,i,p,n,l,temp,tmp2,tmp3,rhocgs,t1,tmp4,
     &   tmp5,tmp6
      real*8 mbig,xcen(3),vcen(3)
c      real*8 v0(3,NMAX),x0(3,NMAX)
      logical test,oldflag,flag1,flag2
      character*1 c1
      character*3 c3,alg(60)
      character*80 infile(3),filename,c80
```

```
      character*150 string
c
c
     ------------------------------------------------------------------
c
      data alg/'MVS','Mvs','mvs','mvs','mvs','BS ','Bs ','
     &  bs ','Bul',
     %  'bul','BS2','Bs2','bs2','Bu2','bu2','RAD','Rad','
     &  rad','RA ',
     %  'ra ','xxx','xxx','xxx','xxx','xxx','xxx','xxx','
     &  xxx','xxx',
     %  'xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx','
     &  xxx','xxx',
     %  'xxx','TES','Tes','tes','Tst','tst','HYB','Hyb','
     &  hyb','HY ',
     %  'hy ','CLO','Clo','clo','CB ','cb ','WID','Wid','
     &  wid','WB ',
     %  'wb '/
c
      rhocgs = AU * AU * AU * K2 / MSUN
      do j = 1, 80
        filename(j:j) = ' '
      end do
      do j = 1, 3
        infile(j)   = filename
```

```
          outfile(j)  = filename
          dumpfile(j) = filename
        end do
        dumpfile(4) = filename
c
c Read in output messages
        inquire (file='message.in', exist=test)
        if (.not.test) then
          write (*,'(/,2a)') ' ERROR: This file is needed to
              start',
     %    ' the integration:  message.in'
          stop
        end if
        open (16, file='message.in', status='old')
  10    read (16,'(i3,1x,i2,1x,a80)',end=20) j,lmem(j),mem(j
     )
        goto 10
  20    close (16)
c
c Read in filenames and check for duplicate filenames
        inquire (file='files.in', exist=test)
        if (.not.test) call mio_err (6,mem(81),lmem(81),mem
           (88),lmem(88),
     %  ' ',1,'files.in',8)
        open (15, file='files.in', status='old')
c
```

```
c Input files
      do j = 1, 3
        read (15,'(a150)') string
        call mio_spl (150,string,nsub,lim)
        infile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim
           (1,1):lim(2,1))
        do k = 1, j - 1
          if (infile(j).eq.infile(k)) call mio_err (6,mem
             (81),lmem(81),
     %       mem(89),lmem(89),infile(j),80,mem(86),lmem(86)
        )
        end do
      end do
c
c Output files
      do j = 1, 3
        read (15,'(a150)') string
        call mio_spl (150,string,nsub,lim)
        outfile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim
           (1,1):lim(2,1))
        do k = 1, j - 1
          if (outfile(j).eq.outfile(k)) call mio_err (6,
             mem(81),
     %       lmem(81),mem(89),lmem(89),outfile(j),80,mem
        (86),lmem(86))
        end do
```

```
          do k = 1, 3
            if (outfile(j).eq.infile(k)) call mio_err (6,mem
               (81),lmem(81),
     %       mem(89),lmem(89),outfile(j),80,mem(86),lmem
          (86))
          end do
        end do
c
c Dump files
        do j = 1, 4
          read (15,'(a150)') string
          call mio_spl (150,string,nsub,lim)
          dumpfile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim
             (1,1):lim(2,1))
          do k = 1, j - 1
            if (dumpfile(j).eq.dumpfile(k)) call mio_err (6,
               mem(81),
     %       lmem(81),mem(89),lmem(89),dumpfile(j),80,mem
          (86),lmem(86))
          end do
          do k = 1, 3
            if (dumpfile(j).eq.infile(k)) call mio_err (6,
               mem(81),
     %       lmem(81),mem(89),lmem(89),dumpfile(j),80,mem
          (86),lmem(86))
          end do
```

```
          do k = 1, 3
            if (dumpfile(j).eq.outfile(k)) call mio_err (6,
               mem(81),
     %        lmem(81),mem(89),lmem(89),dumpfile(j),80,mem
          (86),lmem(86))
          end do
        end do
        close (15)
c
c Find out if this is an old integration (i.e. does the
    restart file exist)
        inquire (file=dumpfile(4), exist=oldflag)
c
c Check if information file exists, and append a
    continuation message
        if (oldflag) then
          inquire (file=outfile(3), exist=test)
          if (.not.test) call mio_err (6,mem(81),lmem(81),
             mem(88),
     %     lmem(88),' ',1,outfile(3),80)
 320      open(23,file=outfile(3),status='old',access='
     append',err=320)
        else
c
c If new integration, check information file doesn't exist
    , and then create it
```

```
          inquire (file=outfile(3), exist=test)
          if (test) call mio_err (6,mem(81),lmem(81),mem(87)
             ,lmem(87),
     %     ' ',1,outfile(3),80)
 410      open(23, file = outfile(3), status = 'new', err
    =410)
        end if
c
c
```

```
c
c  READ  IN  INTEGRATION  PARAMETERS
c
c Check if the file containing integration parameters
   exists, and open it
        filename = infile(3)
        if (oldflag) filename = dumpfile(3)
        inquire (file=filename, exist=test)
        if (.not.test) call mio_err (23,mem(81),lmem(81),mem
           (88),lmem(88),
     %  ' ',1,filename,80)
  30   open  (13, file=filename, status='old', err=30)
c
c Read integration parameters
        lineno = 0
```

```
      do j = 1, 26
40      lineno = lineno + 1
        read (13,'(a150)') string
        if (string(1:1).eq.')') goto 40
        call mio_spl (150,string,nsub,lim)
        c80(1:3) = '   '
        c80 = string(lim(1,nsub):lim(2,nsub))
        if (j.eq.1) then
          algor = 0
           do k = 1, 60
             if (c80(1:3).eq.alg(k)) algor = (k + 4) / 5
          end do
          if (algor.eq.0) call mio_err (23,mem(81),lmem
             (81),mem(98),
     %       lmem(98),c80(lim(1,nsub):lim(2,nsub)),lim(2,
        nsub)-
     %       lim(1,nsub)+1,mem(85),lmem(85))
        end if
        if (j.eq.2) read (c80,*,err=661) tstart
        if (j.eq.3) read (c80,*,err=661) tstop
        if (j.eq.4) read (c80,*,err=661) dtout
        if (j.eq.5) read (c80,*,err=661) h0
        if (j.eq.6) read (c80,*,err=661) tol
        c1 = c80(1:1)
        if (j.eq.7.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(1) =
           1
```

```
if (j.eq.8.and.(c1.eq.'n'.or.c1.eq.'N')) opt(2) =
   0
if (j.eq.9.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(2) =
   2
if (j.eq.10.and.(c1.eq.'d'.or.c1.eq.'D')) opt(3) =
    0
if (j.eq.11.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(3) =
    opt(3) + 2
if (j.eq.12) then
  if(c1.eq.'l'.or.c1.eq.'L') then
    opt(4) = 1
  else if (j.eq.12.and.(c1.eq.'m'.or.c1.eq.'M'))
     then
    opt(4) = 2
  else if (j.eq.12.and.(c1.eq.'h'.or.c1.eq.'H'))
     then
    opt(4) = 3
  else
    goto 661
  end if
end if
if (j.eq.15.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(8) =
    1
if (j.eq.16) read (c80,*,err=661) rmax
if (j.eq.17) read (c80,*,err=661) rcen
if (j.eq.18) read (c80,*,err=661) m(1)
```

```
            if (j.eq.19) read (c80,*,err=661) jcen(1)
            if (j.eq.20) read (c80,*,err=661) jcen(2)
            if (j.eq.21) read (c80,*,err=661) jcen(3)
            if (j.eq.24) read (c80,*,err=661) cefac
            if (j.eq.25) read (c80,*,err=661) ndump
            if (j.eq.26) read (c80,*,err=661) nfun
          end do
          h0 = abs(h0)
          tol = abs(tol)
          rmax = abs(rmax)
          rcen = abs(rcen)
          cefac = abs(cefac)
          close (13)
c
c Change quantities for central object to suitable units
         m(1) = abs(m(1)) * K2
          jcen(1) = jcen(1) * rcen * rcen
          jcen(2) = jcen(2) * rcen * rcen * rcen * rcen
          jcen(3) = jcen(3) * rcen * rcen * rcen * rcen * rcen
              * rcen
          s(1,1) = 0.d0
          s(2,1) = 0.d0
          s(3,1) = 0.d0
c
c Make sure that RCEN isn't too small, since it is used to
     scale the output
```

```
c (Minimum value corresponds to a central body with
   density 100g/cm^3).
      temp = 1.1235d-3 * m(1) ** .3333333333333333d0
      if (rcen.lt.temp) then
        rcen = temp
        write (13,'(/,2a)') mem(121)(1:lmem(121)),mem(131)
           (1:lmem(131))
      end if
c
c
   ------------------------------------------------------------------------


c
c  READ IN DATA FOR BIG AND SMALL BODIES
c
      mbig = m(1)
      xcen(1) = 0.d0
      xcen(2) = 0.d0
      xcen(3) = 0.d0
      vcen(1) = 0.d0
      vcen(2) = 0.d0
      vcen(3) = 0.d0
c
      nbod = 1
      do j = 1, 2
        if (j.eq.2) nbig = nbod
```

```
c
c Check if the file containing data for Big bodies exists,
    and open it
        filename = infile(j)
        if (oldflag) filename = dumpfile(j)
        inquire (file=filename, exist=test)
        if (.not.test) call mio_err (23,mem(81),lmem(81),
           mem(88),
     %    lmem(88),' ',1,filename,80)
 110    open (11, file=filename, status='old', err=110)
c
c Read data style
 120    read (11,'(a150)') string
        if (string(1:1).eq.')') goto 120
        call mio_spl (150,string,nsub,lim)
        c3 = string(lim(1,nsub):(lim(1,nsub)+2))
        if (c3.eq.'Car'.or.c3.eq.'car'.or.c3.eq.'CAR')
           then
          informat = 1
        else if (c3.eq.'Ast'.or.c3.eq.'ast'.or.c3.eq.'AST
           ') then
          informat = 2
        else if (c3.eq.'Com'.or.c3.eq.'com'.or.c3.eq.'COM
           ') then
          informat = 3
        else if (c3.eq.'Bas'.or.c3.eq.'bas'.or.c3.eq.'BAS
```

```
            ') then
            informat = 4
          else
            call mio_err (23,mem(81),lmem(81),mem(91),lmem
               (91),' ',1,
     %       mem(82+j),lmem(82+j))
          end if
c
c Read epoch of Big bodies
          if (j.eq.1) then
 125        read (11,'(a150)') string
            if (string(1:1).eq.')') goto 125
            call mio_spl (150,string,nsub,lim)
            read (string(lim(1,nsub):lim(2,nsub)),*,err=667)
                time
          end if
c
c Read information for each object
 130      read (11,'(a)',end=140) string
          if (string(1:1).eq.')') goto 130
          call mio_spl (150,string,nsub,lim)
          if (lim(1,1).eq.-1) goto 140
c
c Determine the name of the object
          nbod = nbod + 1
          if (nbod.gt.NMAX) call mio_err (23,mem(81),lmem
```

```
            (81),mem(90),
     %     lmem(90),' ',1,mem(82),lmem(82))
c
          if ((lim(2,1)-lim(1,1)).gt.7) then
            write (23,'(/,3a)') mem(121)(1:lmem(121)),
     %       mem(122)(1:lmem(122)),string( lim(1,1):lim
          (2,1) )
          end if
          id(nbod) = string( lim(1,1):min(7+lim(1,1),lim
             (2,1)) )
c Check if another object has the same name
          do k = 1, nbod - 1
            if (id(k).eq.id(nbod)) call mio_err (23,mem(81),
               lmem(81),
     %       mem(103),lmem(103),id(nbod),8,' ',1)
          end do
c
c Default values of mass, close-encounter limit, density
    etc.
          m(nbod) = 0.d0
          rceh(nbod) = 1.d0
          rho(nbod) = rhocgs
          epoch(nbod) = time
          do k = 1, 4
            ngf(k,nbod) = 0.d0
          end do
```

```
c
c Read values of mass, close-encounter limit, density etc.
        do k = 3, nsub, 2
          c80 = string(lim(1,k-1):lim(2,k-1))
          read (string(lim(1,k):lim(2,k)),*,err=666) temp
          if (c80(1:1).eq.'m'.or.c80(1:1).eq.'M') then
            m(nbod) = temp * K2
          else if (c80(1:1).eq.'r'.or.c80(1:1).eq.'R')
             then
            rceh(nbod) = temp
          else if (c80(1:1).eq.'d'.or.c80(1:1).eq.'D')
             then
            rho(nbod) = temp * rhocgs
          else if (m(nbod).lt.0.or.rceh(nbod).lt.0.or.rho(
             nbod).lt.0)
     %       then
            call mio_err (23,mem(81),lmem(81),mem(97),lmem
               (97),id(nbod),
     %         8,mem(82+j),lmem(82+j))
          else if (c80(1:2).eq.'ep'.or.c80(1:2).eq.'EP'.or
             .c80(1:2)
     %         .eq.'Ep') then
            epoch (nbod) = temp
          else if (c80(1:2).eq.'a1'.or.c80(1:2).eq.'A1')
             then
            ngf (1,nbod) = temp
```

```
            else if (c80(1:2).eq.'a2'.or.c80(1:2).eq.'A2')
               then
              ngf (2,nbod) = temp
            else if (c80(1:2).eq.'a3'.or.c80(1:2).eq.'A3')
               then
              ngf (3,nbod) = temp
            else if (c80(1:1).eq.'b'.or.c80(1:1).eq.'B')
               then
              ngf (4,nbod) = temp
            else
              goto 666
            end if
          end do
          if (j.eq.1) mbig = mbig  +  m(nbod)
c
c If using barycentric orbit for small body, it must have
   the same epoch
c as the big bodies
          if (informat.eq.4.and.epoch(nbod).ne.time) then
            write (*,'(/,2a)') ' ERROR: Programme terminated
               . See',
     %       ' information file for details.'
            write (23,'(/,3a)') ' ERROR: Big and small
               bodies must',
     %       ' all have the same epoch if reading in
          barycentric',
```

```
     %       ' elements.'
             stop
           end if
c
c If required, read Cartesian coordinates, velocities and
   spins of the bodies
           jtmp = 100
 135       read (11,'(a150)',end=666) string
           if (string(1:1).eq.')') goto 135
           backspace 11
           if (informat.eq.1) then
             read (11,*,err=666) x(1,nbod),x(2,nbod),x(3,nbod
                ),
     %         v(1,nbod),v(2,nbod),v(3,nbod),s(1,nbod),s(2,
           nbod),s(3,nbod)
           else
             read (11,*,err=666) a,e,i,p,n,l,s(1,nbod),s(2,
                nbod),
     %         s(3,nbod)
             i = i * DR
             p = (p + n) * DR
             n = n * DR
             temp = m(nbod) + m(1)
c
c If using barycentric elements for small bodies:
             if (informat.eq.4.and.j.eq.2) temp = mbig + m(
```

```
     nbod)
c
c Alternatively, read Cometary or asteroidal elements
c          if (informat.eq.3) then
c            q = a
c            a = q / (1.d0 - e)
c            l = mod (sqrt(temp/(abs(a*a*a))) * (epoch(
   nbod) - l), TWOPI)
c          else
c            q = a * (1.d0 - e)
c            l = l * DR
c          end if
c          if (algor.eq.11.and.nbod.ne.2) temp = temp + m
   (2)
c          call mco_el2x (temp,q,e,i,p,n,l,x(1,nbod),x(2,
   nbod),x(3,nbod),
c     %       v(1,nbod),v(2,nbod),v(3,nbod))
c
c If using barycentric elements for small bodies convert
   to heliocentric elements
         if (informat.eq.4.and.j.eq.2) then
           x(1,nbod) = x(1,nbod) - xcen(1)
           x(2,nbod) = x(2,nbod) - xcen(2)
           x(3,nbod) = x(3,nbod) - xcen(3)
           v(1,nbod) = v(1,nbod) - vcen(1)
           v(2,nbod) = v(2,nbod) - vcen(2)
```

```
            v(3,nbod) = v(3,nbod)  -  vcen(3)
          end if
        end if
c
c Update barycentric position and velocity of central body
        if (j.eq.1) then
          xcen(1) = xcen(1)  -  m(nbod) * x(1,nbod)
          xcen(2) = xcen(2)  -  m(nbod) * x(2,nbod)
          xcen(3) = xcen(3)  -  m(nbod) * x(3,nbod)
          vcen(1) = vcen(1)  -  m(nbod) * v(1,nbod)
          vcen(2) = vcen(2)  -  m(nbod) * v(2,nbod)
          vcen(3) = vcen(3)  -  m(nbod) * v(3,nbod)
        end if
c         nbod start from 2
        s(1,nbod) = s(1,nbod)
        s(2,nbod) = s(2,nbod)
        s(3,nbod) = s(3,nbod)
c         s(1,nbod) = s(1,nbod) * K2
c         s(2,nbod) = s(2,nbod) * K2
c         s(3,nbod) = s(3,nbod) * K2
c
        goto 130
 140    close (11)
c
c Calculate barycentric position and velocity of central
   body
```

```
          if (j.eq.1) then
            xcen(1) = xcen(1) / mbig
            xcen(2) = xcen(2) / mbig
            xcen(3) = xcen(3) / mbig
            vcen(1) = vcen(1) / mbig
            vcen(2) = vcen(2) / mbig
            vcen(3) = vcen(3) / mbig
          end if
        end do
c
c Set non-gravitational-forces flag, NGFLAG
        ngflag = 0
        do j = 2, nbod
          if (ngf(1,j).ne.0.or.ngf(2,j).ne.0.or.ngf(3,j).ne
             .0) then
            if (ngflag.eq.0) ngflag = 1
            if (ngflag.eq.2) ngflag = 3
          else if (ngf(4,j).ne.0) then
            if (ngflag.eq.0) ngflag = 2
            if (ngflag.eq.1) ngflag = 3
          end if
        end do
c
c
```

```
c
c   IF  CONTINUING  AN  OLD  INTEGRATION
c
      if (oldflag) then
        if (opt(3).eq.1) then
          call mio_jd2y (time,year,month,t1)
          write (23,'(/,a,i10,i2,f8.5,/)') mem(62)(1:lmem
             (62)),year,
     %       month,t1
        else if (opt(3).eq.3) then
          t1 = (time - tstart) / 365.25d0
          write (23,'(/,a,f18.7,a,/)') mem(62)(1:lmem(62))
             ,t1,
     %       mem(2)(1:lmem(2))
        else
          if (opt(3).eq.0) t1 = time
          if (opt(3).eq.2) t1 = time - tstart
          write (23,'(/,a,f18.5,a,/)') mem(62)(1:lmem(62))
             ,t1,
     %       mem(1)(1:lmem(1))
        end if
c
c Read in energy and angular momentum variables, and
   convert to internal units
 330    open (35, file=dumpfile(4), status='old', err=330)
          read (35,*) opflag
```

```
            read (35,*) en(1),am(1),en(3),am(3)
            en(1) = en(1) * K2
            en(3) = en(3) * K2
            am(1) = am(1) * K2
            am(3) = am(3) * K2
            read (35,*) s(1,1),s(2,1),s(3,1)
            s(1,1) = s(1,1) * K2
            s(2,1) = s(2,1) * K2
            s(3,1) = s(3,1) * K2
          close (35)
          if (opflag.eq.0) opflag = 1
c
c
     ------------------------------------------------------------------------


c
c   IF  STARTING  A  NEW  INTEGRATION
c
        else
          opflag = -2
c
c Write integration parameters to information file
          write (23,'(/,a)') mem(11)(1:lmem(11))
          write (23,'(a)') mem(12)(1:lmem(12))
          j = algor + 13
          write (23,'(/,2a)') mem(13)(1:lmem(13)),mem(j)(1:
```

```
      lmem(j))
    if (tstart.ge.1.d11.or.tstart.le.-1.d10) then
      write (23,'(/,a,1p,e19.13,a)') mem(26)(1:lmem
        (26)),tstart,
%     mem(1)(1:lmem(1))
    else
      write (23,'(/,a,f19.7,a)') mem(26)(1:lmem(26)),
        tstart,
%     mem(1)(1:lmem(1))
    end if
    if (tstop.ge.1.d11.or.tstop.le.-1.d10) then
      write (23,'(a,1p,e19.13)') mem(27)(1:lmem(27)),
        tstop
    else
      write (23,'(a,f19.7)') mem(27)(1:lmem(27)),tstop
    end if
    write (23,'(a,f15.3)') mem(28)(1:lmem(28)),dtout
    if (opt(4).eq.1) write (23,'(2a)') mem(40)(1:lmem
      (40)),
%     mem(7)(1:lmem(7))
    if (opt(4).eq.2) write (23,'(2a)') mem(40)(1:lmem
      (40)),
%     mem(8)(1:lmem(8))
    if (opt(4).eq.3) write (23,'(2a)') mem(40)(1:lmem
      (40)),
%     mem(9)(1:lmem(9))
```

```
c
      write (23,'(/,a,f10.3,a)') mem(30)(1:lmem(30)),h0,
     %   mem(1)(1:lmem(1))
      write (23,'(a,1p1e10.4)') mem(31)(1:lmem(31)),tol
      write (23,'(a,1p1e10.4,a)') mem(32)(1:lmem(32)),m
         (1)/K2,
     %   mem(3)(1:lmem(3))
      write (23,'(a,1p1e11.4)') mem(33)(1:lmem(33)),jcen
         (1)/rcen**2
      write (23,'(a,1p1e11.4)') mem(34)(1:lmem(34)),jcen
         (2)/rcen**4
      write (23,'(a,1p1e11.4)') mem(35)(1:lmem(35)),jcen
         (3)/rcen**6
      write (23,'(a,1p1e10.4,a)') mem(36)(1:lmem(36)),
         rmax,
     %   mem (4)(1:lmem(4))
      write (23,'(a,1p1e10.4,a)') mem(37)(1:lmem(37)),
         rcen,
     %   mem (4)(1:lmem(4))
c
      itmp = 5
      if (opt(2).eq.1.or.opt(2).eq.2) itmp = 6
      write (23,'(/,2a)') mem(41)(1:lmem(41)),mem(itmp)
         (1:lmem(itmp))
      itmp = 5
      if (opt(2).eq.2) itmp = 6
```

```
          write (23,'(2a)') mem(42)(1:lmem(42)),mem(itmp)(1:
             lmem(itmp))
          itmp = 5
          if (opt(7).eq.1) itmp = 6
          write (23,'(2a)') mem(45)(1:lmem(45)),mem(itmp)(1:
             lmem(itmp))
          itmp = 5
          if (opt(8).eq.1) itmp = 6
          write (23,'(2a)') mem(46)(1:lmem(46)),mem(itmp)(1:
             lmem(itmp))
c
c Check that element and close-encounter files don't exist
   , and create them
          do j = 1, 2
            inquire (file=outfile(j), exist=test)
            if (test) call mio_err (23,mem(81),lmem(81),mem
               (87),lmem(87),
     %       ' ',1,outfile(j),80)
 430        open  (20+j, file=outfile(j), status='new', err
    =430)
            close (20+j)
          end do
c
c Check that dump files don't exist, and then create them
          do j = 1, 4
            inquire (file=dumpfile(j), exist=test)
```

```
            if (test) call mio_err (23,mem(81),lmem(81),mem
               (87),lmem(87),
     %        ' ',1,dumpfile(j),80)
 450        open (30+j, file=dumpfile(j), status='new', err
    =450)
            close (30+j)
          end do
c
c Write number of Big bodies and Small bodies to
   information file
          write (23,'(/,a,i4)') mem(38)(1:lmem(38)), nbig -
             1
          write (23,'(a,i4)') mem(39)(1:lmem(39)), nbod -
             nbig
c
c Calculate initial energy and angular momentum and write
   to information file
          s(1,1) = 0.d0
          s(2,1) = 0.d0
          s(3,1) = 0.d0
          call mxx_en (jcen,nbod,nbig,m,x,v,s,en(1),am(1))
          write (23,'(//,a)') mem(51)(1:lmem(51))
          write (23,'(a)')    mem(52)(1:lmem(52))
          write (23,'(/,a,1p1e12.5,a)') mem(53)(1:lmem(53)),
             en(1)/K2,
     %    mem(72)(1:lmem(72))
```

```
          write (23,'(a,1p1e12.5,a)')    mem(54)(1:lmem(54)),
             am(1)/K2,
     %     mem(73)(1:lmem(73))
c
c Initialize lost energy and angular momentum
          en(3) = 0.d0
          am(3) = 0.d0
c
c Write warning messages if necessary
          if (tstop.lt.tstart) write (23,'(/,2a)') mem(121)
             (1:lmem(121)),
     %     mem(123)(1:lmem(123))
          if (nbig.le.0) write (23,'(/,2a)') mem(121)(1:lmem
             (121)),
     %     mem(124)(1:lmem(124))
          if (nbig.eq.nbod) write (23,'(/,2a)') mem(121)(1:
             lmem(121)),
     %     mem(125)(1:lmem(125))
        end if
c
c
      ------------------------------------------------------------------
c
c  CHECK FOR ATTEMPTS TO DO INCOMPATIBLE THINGS
c
```

```
c If using close-binary algorithm, set radius of central
   body to be no less
c than the periastron of binary star.
c       if (algor.eq.11) then
c          temp = m(1) + m(2)
c          call mco_x2el (temp,x(1,2),x(2,2),x(3,2),v(1,2),v
   (2,2),v(3,2),
c     %     a,tmp2,tmp3,tmp4,tmp5,tmp6)
c          rcen = max (rcen, a)
c       end if
c
c Check if non-grav forces are being used with an
   incompatible algorithm
        if (ngflag.ne.0.and.(algor.eq.3.or.algor.eq.11.or.
           algor.eq.12))
     %  call mio_err (23,mem(81),lmem(81),mem(92),lmem(92)
           ,' ',1,
     %  mem(85),lmem(85))
c
c Check if user-defined force routine is being used with
   wrong algorithm
        if (opt(8).eq.1.and.(algor.eq.11.or.algor.eq.12))
           call mio_err
     %  (23,mem(81),lmem(81),mem(93),lmem(93),' ',1,mem
           (85),lmem(85))
c
```

```
c Check whether MVS is being used to integrate massive
   Small bodies,
c or whether massive Small bodies have different epochs
   than Big bodies.
      flag1 = .false.
      flag2 = .false.
      do j = nbig + 1, nbod
        if (m(j).ne.0) then
          if (algor.eq.1) call mio_err (23,mem(81),lmem
             (81),mem(94),
     %       lmem(94),' ',1,mem(85),lmem(85))
          flag1 = .true.
        end if
        if (epoch(j).ne.time) flag2 = .true.
      end do
      if (flag1.and.flag2) call mio_err (23,mem(81),lmem
         (81),mem(95),
     %    lmem(95),' ',1,mem(84),lmem(84))
c
c Check if central oblateness is being used with close-
   binary algorithm
      if (algor.eq.11.and.(jcen(1).ne.0.or.jcen(2).ne.0.or
         .jcen(3)
     %  .ne.0)) call mio_err (23,mem(81),lmem(81),mem(102)
         ,lmem(102),
     %  ' ',1,mem(85),lmem(85))
```

```
c
c Check whether RCEN > RMAX or RMAX/RCEN is very large
      if (rcen.gt.rmax) call mio_err (23,mem(81),lmem(81),
         mem(105),
     %  lmem(105),' ',1,mem(85),lmem(85))
      if (rmax/rcen.ge.1.d12) write (23,'(/,2a,/a)')
     %  mem(121)(1:lmem(121)),mem(106)(1:lmem(106)),mem
        (85)(1:lmem(85))
      close (23)
      return
c
c Error reading from the input file containing integration
    parameters
 661  write (c3,'(i3)') lineno
      call mio_err (23,mem(81),lmem(81),mem(99),lmem(99),
         c3,3,
     %  mem(85),lmem(85))
c
c Error reading from the input file for Big or Small
   bodies
 666  call mio_err (23,mem(81),lmem(81),mem(100),lmem(100)
    ,id(nbod),8,
     %  mem(82+j),lmem(82+j))
c
c Error reading epoch of Big bodies
 667  call mio_err (23,mem(81),lmem(81),mem(101),lmem(101)
```

```
     ,' ',1,
     %  mem(83),lmem(83))
c
c
     ------------------------------------------------------------------------

c
      end
c
c
     %%%%
```

```
c The output variables are:
c  r = the radial distance
c  theta = polar angle
c  phi = azimuthal angle
c  fv = 1 / [1 + 2(ke/be)^2], where be and ke are the
   object's binding and
c                              kinetic energies. (Note that
    0 < fv < 1).
c  vtheta = polar angle of velocity vector
c  vphi = azimuthal angle of the velocity vector
c
c If this is the first output (OPFLAG = -1), or the first
   output since the
c number of the objects or their masses have changed (
   OPFLAG = 1), then
c the names, masses and spin components of all the objects
    are also output.
c
c N.B. Each object's distance must lie between RCEN < R <
   RMAX
c ===
c
c
   ------------------------------------------------------------------------


c
```

```
      subroutine mio_out (time,jcen,rcen,rmax,nbod,nbig,m,
         xh,vh,s,rho,
     %   stat,id,opt,opflag,algor,outfile)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod), opt(8), opflag,
         algor
      real*8 time,jcen(3),rcen,rmax,m(nbod),xh(3,nbod),vh
         (3,nbod)
      real*8 s(3,nbod),rho(nbod)
      character*80 outfile
      character*8 id(nbod)
c
c Local
      integer k, len, nchar
      real*8 rhocgs,k_2,rfac,rcen_2,fr,fv,theta,phi,vtheta
         ,vphi
      real*8 stheta,sphi
      character*80 header,c(NMAX)
      character*8 mio_fl2c,mio_re2c
      real*8 mio_c2re
      character*5 fout
c
```

```
c
c     ------------------------------------------------------------------
c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
c
c Scaling factor (maximum possible range) for distances
      rfac = log10 (rmax / rcen)
c
c Create the format list, FOUT, used when outputting the
   orbital elements
      if (opt(4).eq.1) nchar = 2
      if (opt(4).eq.2) nchar = 4
      if (opt(4).eq.3) nchar = 7
      len = 3  +  8 * nchar
      fout(1:5) = '(a  )'
      if (len.lt.10) write (fout(3:3),'(i1)') len
      if (len.ge.10) write (fout(3:4),'(i2)') len
c
c Open the orbital-elements output file
  10  open (21, file=outfile, status='old', access='append
     ', err=10)
c
c
```

```
c
c  SPECIAL  OUTPUT  PROCEDURE
c
c If this is a new integration or a complete output is
   required (e.g. because
c the number of objects has changed), then output object
   details & parameters.
      if (opflag.eq.-1.or.opflag.eq.1) then
c
c Compose a header line with time, number of objects and
   relevant parameters
        header(1:8)   = mio_fl2c (time)
        header(9:16)  = mio_re2c (dble(nbig - 1),   0.d0,
           11239423.99d0)
        header(12:19) = mio_re2c (dble(nbod - nbig),0.d0,
           11239423.99d0)
        header(15:22) = mio_fl2c (m(1) * k_2)
        header(23:30) = mio_fl2c (jcen(1) * rcen_2)
        header(31:38) = mio_fl2c (jcen(2) * rcen_2 *
           rcen_2)
        header(39:46) = mio_fl2c (jcen(3) * rcen_2 *
           rcen_2 * rcen_2)
        header(47:54) = mio_fl2c (rcen)
        header(55:62) = mio_fl2c (rmax)
```

```
c
c For each object, compress its index number, name, mass,
   spin components
c and density (some of these need to be converted to
   normal units).
        do k = 2, nbod
          c(k)(1:8) = mio_re2c (dble(k - 1), 0.d0,
             11239423.99d0)
          c(k)(4:11) = id(k)
          c(k)(12:19) = mio_fl2c (m(k) * k_2)
c          c(k)(20:27) = mio_fl2c (s(1,k))
c          c(k)(28:35) = mio_fl2c (s(2,k))
c          c(k)(36:43) = mio_fl2c (s(3,k))
          c(k)(44:51) = mio_fl2c (rho(k) / rhocgs)
        end do
c
c Write compressed output to file
        write (21,'(a1,a2,i2,a62,i1)') char(12),'6a',algor
           ,header(1:62),
     %     opt(4)
        do k = 2, nbod
          write (21,'(a51)') c(k)(1:51)
        end do
      end if
c
c
```

```
c
c  NORMAL  OUTPUT  PROCEDURE
c
c Compose a header line containing the time and number of
   objects
      header(1:8)   = mio_fl2c (time)
      header(9:16)  = mio_re2c (dble(nbig - 1),    0.d0,
         11239423.99d0)
      header(12:19) = mio_re2c (dble(nbod - nbig), 0.d0,
         11239423.99d0)
c
c Calculate output variables for each body and convert to
   compressed format
      do k = 2, nbod
         call mco_x2ov_s (rcen,rmax,m(1),m(k),xh(1,k),xh(2,
            k),xh(3,k),
     %     vh(1,k),vh(2,k),vh(3,k),s(1,k),s(2,k),s(3,k),fr,
         theta,
     %     phi,fv,vtheta,vphi,stheta,sphi)
c
c Object's index number and output variables
         c(k)(1:8) = mio_re2c (dble(k - 1), 0.d0,
            11239423.99d0)
c         back conversion successful. key lies in xmin-xmax
```

```
      range
          c(k)(4:11)                    = mio_re2c (fr,      0.
             d0, rfac)
          c(k)(4+  nchar:11+  nchar) = mio_re2c (theta, 0.d0
             , PI)
          c(k)(4+2*nchar:11+2*nchar) = mio_re2c (phi,     0.
             d0, TWOPI)
          c(k)(4+3*nchar:11+3*nchar) = mio_re2c (fv,      0.
             d0, 1.d0)
          c(k)(4+4*nchar:11+4*nchar) = mio_re2c (vtheta, 0.
             d0, PI)
          c(k)(4+5*nchar:11+5*nchar) = mio_re2c (vphi,    0.
             d0, TWOPI)
          c(k)(4+6*nchar:11+6*nchar) = mio_re2c (stheta, 0.
             d0, PI)
          c(k)(4+7*nchar:11+7*nchar) = mio_re2c (sphi,    0.
             d0, TWOPI)
        end do
c
c Write compressed output to file
        write (21,'(a1,a2,a14)') char(12),'6b',header(1:14)
        do k = 2, nbod
          write (21,fout) c(k)(1:len)
        end do
c
        close (21)
```

```
      opflag = 0
c
c
c     ------------------------------------------------------------------
c
      return
      end
c
c
c%%%
```

```
     digit is converted
c to an ASCII character, such that 0 -> character 32, 1 ->
     character 33...
c and 223 -> character 255.
c
c ASCII characters 0 - 31 (CTRL characters) are not used,
    because they
c cause problems when using some operating systems.
c
c
    ------------------------------------------------------------------


c
      function mio_re2c (x,xmin,xmax)
c
      implicit none
c
c Input/output
      real*8 x,xmin,xmax
      character*8 mio_re2c
c
c Local
      integer j
      real*8 y,z
c
c
```

```
c
      mio_re2c(1:8) = '        '
      y = (x - xmin) / (xmax - xmin)
c
      if (y.ge.1) then
        do j = 1, 8
          mio_re2c(j:j) = char(255)
        end do
      else if (y.gt.0) then
        z = y
        do j = 1, 8
          z = mod(z, 1.d0) * 224.d0
          mio_re2c(j:j) = char(int(z) + 32)
        end do
      end if
c
c
```

```
c
      return
      end
c
c
```

```
%%%
```

```
c
      function mio_c2re (c,xmin,xmax,nchar)
c
      implicit none
c
c Input/output
      integer nchar
      real*8 xmin,xmax,mio_c2re
      character*8 c
c
c Local
      integer j
      real*8 y
c
c
-----------------------------------------------------------------------
c
      y = 0
      do j = nchar, 1, -1
        y = (y + dble(mod(ichar(c(j:j)) + 256, 256) - 32))
     .      / 224.d0
      end do
c
      mio_c2re = xmin  +  y * (xmax - xmin)
```

```
c
c
c     ------------------------------------------------------------------
c
c
c          return
c          end
c
c
c     %%%%
```

```
c returned in the array DELIMIT.
c Substrings are those which are separated by spaces or
   the = symbol.
c
c
   ------------------------------------------------------------------


c
      subroutine mio_spl (len,string,nsub,delimit)
c
      implicit none
c
c Input/Output
      integer len,nsub,delimit(2,100)
      character*1 string(len)
c
c Local
      integer j,k
      character*1 c
c
c
   ------------------------------------------------------------------


c
      nsub = 0
      j = 0
```

```
      c = ' '
      delimit(1,1) = -1
c
c Find the start of string
   10 j = j + 1
      if (j.gt.len) goto 99
      c = string(j)
      if (c.eq.' '.or.c.eq.'=') goto 10
c
c Find the end of string
      k = j
   20 k = k + 1
      if (k.gt.len) goto 30
      c = string(k)
      if (c.ne.' '.and.c.ne.'=') goto 20
c
c Store details for this string
   30 nsub = nsub + 1
      delimit(1,nsub) = j
      delimit(2,nsub) = k - 1
c
      if (k.lt.len) then
        j = k
        goto 10
      end if
c
```

```
   99 continue
c
c
c ------------------------------------------------------------------------------

c
      return
      end
c
c
c %%%%
```

```
     = 1 on exit,
c otherwise EJFLAG = 0.
c
c Also updates the values of EN(3) and AM(3)---the change
    in energy and
c angular momentum due to collisions and ejections.
c
c
c N.B. All coordinates must be with respect to the central
     body!!
c ===
c
c
      ------------------------------------------------------------------------


c
      subroutine mxx_ejec (time,tstart,rmax,en,am,jcen,i0,
     %   nbod,nbig,m,x,
     %   v,s,stat,id,opt,ejflag,outfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i0, nbod, nbig, stat(nbod), opt(8), ejflag,
     %   lmem(NMESS)
```

```
      real*8 time, tstart, rmax, en(3), am(3), jcen(3)
      real*8 m(nbod), x(3,nbod), v(3,nbod), s(3,nbod)
      character*80 outfile, mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer j, year, month
      real*8 r2,rmax2,t1,e,l
      character*38 flost
      character*6 tstring
c
c
     ------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
      ejflag = 0
      rmax2 = rmax * rmax
c
c Calculate initial energy and angular momentum
      call mxx_en (jcen,nbod,nbig,m,x,v,s,e,l)
c
c Flag each object which is ejected, and set its mass to
     zero
      do j = i0, nbod
        r2 = x(1,j)*x(1,j) + x(2,j)*x(2,j) + x(3,j)*x(3,j)
```

```
          if (r2.gt.rmax2) then
            ejflag = 1
            stat(j) = -3
            m(j) = 0.d0
            s(1,j) = 0.d0
            s(2,j) = 0.d0
            s(3,j) = 0.d0
c
c Write message to information file
  20        open (23,file=outfile,status='old',access='
     append',err=20)
            if (opt(3).eq.1) then
              call mio_jd2y (time,year,month,t1)
              flost = '(1x,a8,a,i10,1x,i2,1x,f8.5)'
              write (23,flost) id(j),mem(68)(1:lmem(68)),
                 year,month,t1
            else
              if (opt(3).eq.3) then
                t1 = (time - tstart) / 365.25d0
                tstring = mem(2)
                flost = '(1x,a8,a,f18.7,a)'
              else
                if (opt(3).eq.0) t1 = time
                if (opt(3).eq.2) t1 = time - tstart
                tstring = mem(1)
                flost = '(1x,a8,a,f18.5,a)'
```

```
            end if
            write (23,flost) id(j),mem(68)(1:lmem(68)),t1,
               tstring
          end if
          close (23)
        end if
      end do
c
c If ejections occurred, update ELOST and LLOST
      if (ejflag.ne.0) then
        call mxx_en (jcen,nbod,nbig,m,x,v,s,en(2),am(2))
        en(3) = en(3) + (e - en(2))
        am(3) = am(3) + (l - am(2))
      end if
c
c
c-----------------------------------------------------------------------
c
      return
      end
c
c
c%%%
```

```
c       MXX_ELIM.FOR    (May 2019)
c
c
%%%
```

```
      real*8 m(nbod), x(3,nbod), v(3,nbod), s(3,nbod)
      real*8 rho(nbod), rceh(nbod), rcrit(nbod), ngf(4,
     &   nbod)
      character*8 id(nbod)
      character*80 outfile, mem(NMESS)
c
c Local
      integer j, k, l, nbigelim, elim(NMAX+1)
c
c
     ------------------------------------------------------------------------
c
c Find out how many objects are to be removed
      nelim = 0
      nbigelim = 0
      do j = 2, nbod
        if (stat(j).lt.0) then
          nelim = nelim + 1
          elim(nelim) = j
          if (j.le.nbig) nbigelim = nbigelim + 1
        end if
      end do
      elim(nelim+1) = nbod + 1
c
c Eliminate unwanted objects
```

```
      do k = 1, nelim
        do j = elim(k) - k + 1, elim(k+1) - k - 1
          l = j + k
          x(1,j) = x(1,l)
          x(2,j) = x(2,l)
          x(3,j) = x(3,l)
          v(1,j) = v(1,l)
          v(2,j) = v(2,l)
          v(3,j) = v(3,l)
          m(j)   = m(l)
          s(1,j) = s(1,l)
          s(2,j) = s(2,l)
          s(3,j) = s(3,l)
          rho(j) = rho(l)
          rceh(j) = rceh(l)
          stat(j) = stat(l)
          id(j) = id(l)
          ngf(1,j) = ngf(1,l)
          ngf(2,j) = ngf(2,l)
          ngf(3,j) = ngf(3,l)
          ngf(4,j) = ngf(4,l)
        end do
      end do
c
c Update total number of bodies and number of Big bodies
      nbod = nbod - nelim
```

```
      nbig = nbig - nbigelim
c
c  If no massive bodies remain, stop the integration
      if (nbig.lt.1) then
 10     open (23,file=outfile,status='old',access='append
     ',err=10)
        write (23,'(2a)') mem(81)(1:lmem(81)),mem(124)(1:
           lmem(124))
        close (23)
        stop
      end if
c
c
c-----------------------------------------------------------------------
c
      return
      end
c
c
c%%%
```

```
%%%
```

```
     nbod),e,l2
c
c Local
      integer j,k,iflag,itmp(8)
      real*8 x(3,NMAX),v(3,NMAX),temp,dx,dy,dz,r2,tmp,ke,
     pe,l(3)
      real*8 r_1,r_2,r_4,r_6,u2,u4,u6,tmp2(4,NMAX)
c
c
c-----------------------------------------------------------------------
c
      ke = 0.d0
      pe = 0.d0
      l(1) = 0.d0
      l(2) = 0.d0
      l(3) = 0.d0
c
c Convert to barycentric coordinates and velocities
      call mco_h2b(temp,jcen,nbod,nbig,temp,m,xh,vh,x,v,
     tmp2,iflag,itmp)
c
c Do the spin angular momenta first (probably the smallest
    terms)
c  account for spin  of central body only
      do j = 1, 1
```

```
c           do j = 1, nbod
              l(1) = l(1) + s(1,j)
              l(2) = l(2) + s(2,j)
              l(3) = l(3) + s(3,j)
            end do
c
c Orbital angular momentum and kinetic energy terms
            do j = 1, nbod
              l(1) = l(1) + m(j)*(x(2,j) * v(3,j) - x(3,j) *
                 v(2,j))
              l(2) = l(2) + m(j)*(x(3,j) * v(1,j) - x(1,j) *
                 v(3,j))
              l(3) = l(3) + m(j)*(x(1,j) * v(2,j) - x(2,j) *
                 v(1,j))
              ke = ke + m(j)*(v(1,j)*v(1,j)+v(2,j)*v(2,j)+v(3,j)
                *v(3,j))
            end do
c
c Potential energy terms due to pairs of bodies
            do j = 2, nbod
              tmp = 0.d0
              do k = j + 1, nbod
                dx = x(1,k) - x(1,j)
                dy = x(2,k) - x(2,j)
                dz = x(3,k) - x(3,j)
                r2 = dx*dx + dy*dy + dz*dz
```

```
            if (r2.ne.0) tmp = tmp + m(k) / dsqrt(r2)
          end do
          pe = pe  -  tmp * m(j)
        end do
c
c Potential energy terms involving the central body
        do j = 2, nbod
          dx = x(1,j) - x(1,1)
          dy = x(2,j) - x(2,1)
          dz = x(3,j) - x(3,1)
          r2 = dx*dx + dy*dy + dz*dz
          if (r2.ne.0) pe = pe  -  m(1) * m(j) / dsqrt(r2)
        end do
c
c Corrections for oblateness
        if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0)
           then
          do j = 2, nbod
            r2 = xh(1,j)*xh(1,j) + xh(2,j)*xh(2,j) + xh(3,j)
               *xh(3,j)
            r_1 = 1.d0 / dsqrt(r2)
            r_2 = r_1 * r_1
            r_4 = r_2 * r_2
            r_6 = r_4 * r_2
            u2 = xh(3,j) * xh(3,j) * r_2
            u4 = u2 * u2
```

```
            u6 = u4 * u2
            pe = pe + m(1) * m(j) * r_1
     %         * (jcen(1) * r_2 * (1.5d0*u2 - 0.5d0)
     %         +  jcen(2) * r_4 * (4.375d0*u4 - 3.75d0*u2 +
        .375d0)
     %         +  jcen(3) * r_6
     %         *(14.4375d0*u6 - 19.6875d0*u4 + 6.5625d0*u2 -
         .3125d0))
           end do
         end if
c
         e = .5d0 * ke  +  pe
         l2 = dsqrt(l(1)*l(1) + l(2)*l(2) + l(3)*l(3))
c
c
      ------------------------------------------------------------------------
c
         return
         end
c
c
      %%%%
```

```
c
c
%%%
```

```
c
c Local
      integer i,j,k,l,m,inc,incarr(9),iy
      real*8 y
      data incarr/1,4,13,40,121,364,1093,3280,9841/
c
c
     ------------------------------------------------------------------


c
      do i = 1, n
        index(i) = i
      end do
c
      m = 0
   10 m = m + 1
      if (incarr(m).lt.n) goto 10
      m = m - 1
c
      do i = m, 1, -1
        inc = incarr(i)
        do j = 1, inc
          do k = inc, n - j, inc
            y = x(j+k)
            iy = index(j+k)
            do l = j + k - inc, j, -inc
```

```
                if (x(l).le.y) goto 20
                x(l+inc) = x(l)
                index(l+inc) = index(l)
              end do
20            x(l+inc) = y
              index(l+inc) = iy
            end do
          end do
        end do
c
c
c-----------------------------------------------------------------------

c
        return
        end
c
c
c%%%
```

```
c
c  Author:  John E. Chambers / Yu-Cian Hong
c
c  Synchronizes the epochs of NBIG Big bodies (having a
     common epoch) and
c  NBOD-NBIG Small bodies (possibly having differing epochs
     ), for an
c  integration using MERCURY.
c  The Small bodies are picked up in order starting with
     the one with epoch
c  furthest from the time, TSTART, at which the main
     integration will begin
c  producing output.
c
c  N.B. The synchronization integrations use Everhart's
     RA15 routine.
c  ---
c
c
c     ------------------------------------------------------------------------


c
      subroutine mxx_sync (time,tstart,h0,tol,jcen,nbod,
     %   nbig,m,x,v,s,
     %   rho,rceh,stat,id,epoch,ngf,opt,ngflag)
c
```

```
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8),stat(nbod)
      real*8 time,tstart,h0,tol,jcen(3),m(nbod),x(3,nbod),
     &   v(3,nbod)
      real*8 s(3,nbod),rceh(nbod),rho(nbod),epoch(nbod),
     &   ngf(4,nbod)
      character*8 id(nbod)
c
c Local
      integer j,k,l,nsml,nsofar,indx(NMAX),itemp,jtemp(
     &   NMAX)
      integer raflag,nce,ice(NMAX),jce(NMAX)
      real*8 temp,epsml(NMAX),rtemp(NMAX)
      real*8 h,hdid,tsmall,rphys(NMAX),rcrit(NMAX)
      character*8 ctemp(NMAX)
      external mfo_all
c
c
------------------------------------------------------------------------
c
c Reorder Small bodies by epoch so that ep(1) is furthest
   from TSTART
```

```
      nsml = nbod - nbig
      do j = nbig + 1, nbod
        epsml(j-nbig) = epoch(j)
      end do
      call mxx_sort (nsml,epsml,indx)
c
      if (abs(epsml(1)-tstart).lt.abs(epsml(nsml)-tstart))
         then
        k = nsml + 1
        do j = 1, nsml / 2
          l = k - j
          temp = epsml(j)
          epsml(j) = epsml (l)
          epsml(l) = temp
          itemp = indx(j)
          indx(j) = indx (l)
          indx(l) = itemp
        end do
      end if
c
      do j = nbig + 1, nbod
        epoch(j) = epsml(j-nbig)
      end do
c
c Reorder the other arrays associated with each Small body
      do k = 1, 3
```

```
        do j = 1, nsml
          rtemp(j) = x(k,j+nbig)
        end do
        do j = 1, nsml
          x(k,j+nbig) = rtemp(indx(j))
        end do
        do j = 1, nsml
          rtemp(j) = v(k,j+nbig)
        end do
        do j = 1, nsml
          v(k,j+nbig) = rtemp(indx(j))
        end do
        do j = 1, nsml
          rtemp(j) = s(k,j+nbig)
        end do
        do j = 1, nsml
          s(k,j+nbig) = rtemp(indx(j))
        end do
      end do
c
      do k = 1, 4
        do j = 1, nsml
          rtemp(j) = ngf(k,j+nbig)
        end do
        do j = 1, nsml
          ngf(k,j+nbig) = rtemp(indx(j))
```

```
        end do
      end do
c
      do j = 1, nsml
        rtemp(j) = m(j+nbig)
      end do
      do j = 1, nsml
        m(j+nbig) = rtemp(indx(j))
      end do
      do j = 1, nsml
        rtemp(j) = rceh(j+nbig)
      end do
      do j = 1, nsml
        rceh(j+nbig) = rtemp(indx(j))
      end do
      do j = 1, nsml
        rtemp(j) = rho(j+nbig)
      end do
      do j = 1, nsml
        rho(j+nbig) = rtemp(indx(j))
      end do
c
      do j = 1, nsml
        ctemp(j) = id(j+nbig)
        jtemp(j) = stat(j+nbig)
      end do
```

```
      do j = 1, nsml
        id(j+nbig) = ctemp(indx(j))
        stat(j+nbig) = jtemp(indx(j))
      end do
c
c Integrate Small bodies up to the same epoch
      h = h0
      tsmall = h0 * 1.d-12
      raflag = 0
c
      do j = nbig + 1, nbod
        nsofar = j - 1
        do while (abs(time-epoch(j)).gt.tsmall)
          temp = epoch(j) - time
          h = sign(max(min(abs(temp),abs(h)),tsmall),temp)
          print *," sync small obody ( add ra15)"
c           call mdt_ra15 (time,h,hdid,tol,jcen,nsofar,nbig
   ,m,x,v,s,rphys,
c     %       rcrit,ngf,stat,raflag,ngflag,opt,nce,ice,jce,
   mfo_all)
          time = time + hdid
        end do
        raflag = 1
      end do
c
c
```

```
c
      return
      end
c
c
%%%
```

```
      subroutine mio_log (time,tstart,en,am,opt,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer lmem(NMESS), opt(8)
      real*8 time, tstart, en(3), am(3)
      character*80 mem(NMESS)
c
c Local
      integer year, month
      real*8 tmp0, tmp1, t1
      character*38 flog
      character*6 tstring
c
c
     ------------------------------------------------------------------------


c
      if (opt(3).eq.0.or.opt(3).eq.2) then
        tstring = mem(1)
        flog = '(1x,a,f14.1,a,2(a,1p1e12.5))'
      else if (opt(3).eq.1) then
        flog = '(1x,a,i10,1x,i2,1x,f4.1,2(a,1p1e12.5))'
      else
```

```
        tstring = mem(2)
        flog = '(1x,a,f14.3,a,2(a,1p1e12.5))'
      end if
c
      tmp0 = 0.d0
      tmp1 = 0.d0
      if (en(1).ne.0) tmp0 = (en(2) + en(3) - en(1)) / abs
     (en(1))
      if (am(1).ne.0) tmp1 = (am(2) + am(3) - am(1)) / abs
     (am(1))
c
      if (opt(3).eq.1) then
        call mio_jd2y (time,year,month,t1)
        write (*,flog) mem(64)(1:lmem(64)), year, month,
     t1,
     %    mem(65)(1:lmem(65)), tmp0,mem(66)(1:lmem(66)),
     tmp1
      else
        if (opt(3).eq.0) t1 = time
        if (opt(3).eq.2) t1 = time - tstart
        if (opt(3).eq.3) t1 = (time - tstart) / 365.25d0
        write (*,flog) mem(63)(1:lmem(63)), t1, tstring,
     %    mem(65)(1:lmem(65)), tmp0, mem(66)(1:lmem(66)),
     tmp1
      end if
c
```

```
c
c     ------------------------------------------------------------------
c
      return
      end
c
c
c%%%
```

```
      function mco_acsh (x)
c
      implicit none
c
c Input/Output
      real*8 x,mco_acsh
c
c
     ------------------------------------------------------------------
c
      if (x.ge.1.d0) then
        mco_acsh = log (x + dsqrt(x*x - 1.d0))
      else
        mco_acsh = 0.d0
      end if
c
c
     ------------------------------------------------------------------
c
      return
      end
c
c
     %%%%
```

```
c
c       MIO_JD2Y.FOR    (ErikSoft  7 July 1999)
c
c
%%%
```

```
c       after 15th October 1582.
c
c
c
      ------------------------------------------------------------------


c
      subroutine mio_jd2y (jd0,year,month,day)
c
      implicit none
c
c Input/Output
      integer year,month
      real*8 jd0,day
c
c Local
      integer i,a,b,c,d,e,g
      real*8 jd,f,temp,x,y,z
c
c
      ------------------------------------------------------------------


c
      if (jd0.le.0) goto 50
c
      jd = jd0 + 0.5d0
```

```
      i = sign( dint(dabs(jd)), jd )
      f = jd - 1.d0*i
c
c If on or after 15th October 1582
      if (i.gt.2299160) then
        temp = (1.d0*i - 1867216.25d0) / 36524.25d0
        a = sign( dint(dabs(temp)), temp )
        temp = .25d0 * a
        b = i + 1 + a - sign( dint(dabs(temp)), temp )
      else
        b = i
      end if
c
      c = b + 1524
      temp = (1.d0*c - 122.1d0) / 365.25d0
      d = sign( dint(dabs(temp)), temp )
      temp = 365.25d0 * d
      e = sign( dint(dabs(temp)), temp )
      temp = (c-e) / 30.6001d0
      g = sign( dint(dabs(temp)), temp )
c
      temp = 30.6001d0 * g
      day = 1.d0*(c-e) + f - 1.d0*sign( dint(dabs(temp)),
     +   temp )
c
      if (g.le.13) month = g - 1
```

```
      if (g.gt.13) month = g - 13
c
      if (month.gt.2) year = d - 4716
      if (month.le.2) year = d - 4715
c
      if (day.gt.32) then
        day = day - 32
        month = month + 1
      end if
c
      if (month.gt.12) then
        month = month - 12
        year = year + 1
      end if
      return
c
 50   continue
c
c Algorithm for negative Julian day numbers (Duffett-Smith
    doesn't work)
      x = jd0 - 2232101.5
      f = x - dint(x)
      if (f.lt.0) f = f + 1.d0
      y = dint(mod(x,1461.d0) + 1461.d0)
      z = dint(mod(y,365.25d0))
      month = int((z + 0.5d0) / 30.61d0)
```

```
      day = dint(z + 1.5d0 - 30.61d0*dble(month)) + f
      month = mod(month + 2, 12) + 1
c
      year = 1399 + int (x / 365.25d0)
      if (x.lt.0) year = year - 1
      if (month.lt.3) year = year + 1
c
c
     ------------------------------------------------------------------
c
      return
      end
c
c
```